
MusicBrainz Picard

Release v2.12.3

Sep 04, 2024

MusicBrainz Picard User Guide by Bob Swift is licensed under CC0 1.0. To view a copy of this license, visit <https://creativecommons.org/publicdomain/zero/1.0>

CONTENTS

1	Introduction	1
1.1	Picard Can...	2
1.2	Picard Cannot...	2
1.3	Limitations	2
2	Contributing to the Project	3
3	Acknowledgements	4
4	Glossary of Terms	6
5	Getting Started	10
5.1	Download & Install Picard	10
5.2	Starting Picard	12
5.3	Main Screen	13
5.4	Status Icons	19
6	Configuration	22
6.1	Screen Setup	22
6.2	Action Options	23
6.3	Option Settings	23
7	Tags & Variables	81
7.1	Basic Tags	81
7.2	Advanced Tags	87
7.3	Basic Variables	90
7.4	File Variables	92
7.5	Advanced Variables	93
7.6	Classical Music Tags	96
7.7	Tags from Plugins	96
7.8	Other Information	100
8	Scripting	101
8.1	Syntax	101
8.2	Metadata Variables	102

9 Scripting Functions	103
9.1 Assignment Functions	103
9.2 Text Functions	107
9.3 Multi-Value Functions	120
9.4 Mathematical Functions	129
9.5 Conditional Functions	132
9.6 Information Functions	146
9.7 Loop Functions	155
9.8 Miscellaneous Functions	156
10 Using Picard	158
10.1 Retrieving Album Information	158
10.2 Matching Files to Tracks	172
10.3 Setting the Cover Art	174
10.4 Saving Updated Files	176
11 Work Flow Recommendations	178
11.1 When the CD is available	178
11.2 When the ripper log file is available	180
11.3 When files are grouped by album	181
11.4 When files are not grouped but have some metadata	182
11.5 When files are not grouped and have little or no existing metadata	183
12 Other Picard Tasks	185
12.1 Attaching a Disc ID to a Release	185
12.2 Submitting Acoustic Fingerprints	188
12.3 Generating tags from file names	193
12.4 Submitting Cluster as a Release	196
13 Option Profiles	200
13.1 How Option Profiles Work	200
13.2 Example of Using Profiles	200
13.3 Managing Option Profiles	201
13.4 Saving Profile Option Settings	204
14 Command and Batch Processing	207
14.1 Executable Commands	209
15 Extending Picard	215
15.1 Plugins	215
15.2 Scripts	216
15.3 Processing Order	220
16 Troubleshooting	222
16.1 General Troubleshooting	222
16.2 Picard won't start	225
16.3 There is no coverart	226
16.4 Tags are not updated or saved	227
16.5 Files are not being saved	228

16.6 Picard just stopped working	228
16.7 macOS shows the app is damaged	229
17 Frequently Asked Questions	230
17.1 Using Picard	230
17.2 File Formats	232
17.3 Configuration	236
18 Tutorials	238
18.1 Writing a File Naming Script	238
18.2 Understanding Acoustic Fingerprinting and AcoustIDs	241
18.3 Handling of multiple release countries	243
18.4 Writing a Plugin	245
18.5 Loading releases with MusicBrainz for Android	250
19 Appendices	263
19.1 Appendix A: Plugins API	263
19.2 Appendix B: Tag Mapping	270
19.3 Appendix C: Command Line Options	294
19.4 Appendix D: Keyboard Shortcuts	296

INTRODUCTION

MusicBrainz Picard is a cross-platform music file tagger. For any people who don't know what this means, here is a quick explanation which can be skipped by those people who already know.

Your music files don't just contain music. They also contain "metadata", consisting of "tags" which consist of a tag name or type and associated data, for example the album or track name, the name of the artist, the record label, the year of issue etc. Unless you rip the music files yourself with a very basic tool, your music files probably already contain some basic metadata, however there are literally hundreds of tags that can be applied to your music if you are interested.

Obviously, if you wanted to you could painstakingly research all this information for each album and track individually, and type the data into a tagging tool, but clearly it makes more sense in this internet connected age for one person to do this for each album and track, to upload that data to a shared database and then for the tagging tool to access that database and use the data to tag the music files. And **that** is what MusicBrainz Picard does.

MusicBrainz is the database, and **Picard** is the tool that tags the music files.

This User Guide is intended to provide comprehensive information related to the use of [MusicBrainz Picard](#) and additionally to make this available in alternate formats, including a PDF version 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. Picard is primarily intended to tag and organize albums containing tracks, guided by the user to the specific release of the album that they have, and then to keep the metadata for these tracks up to date as users around the world enhance the quality of the MusicBrainz data associated with that particular release and track; Picard does this very well indeed. However, it is not intended to automatically organize your collection of thousands of random music files, and if this is what you are hoping for then 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 Picard Can...

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

...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](#).

1.2 Picard Cannot...

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

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

1.3 Limitations

File Formats

Picard currently supports most music file formats, with Matroska (.mka) being one notable exception. Microsoft WAVE (.wav) files can be fingerprinted and renamed and can be tagged using ID3v2 tags, but this is not supported by all playback software. In addition, Picard does not support writing custom tags for all formats.

The [Picard Tag Mappings](#) section 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 noticeable 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.

CONTRIBUTING TO THE PROJECT

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.

Further high quality contributions are welcomed from all Picard users wanting to be part of the open source community that creates and maintains this valuable music tool. Even if you cannot write code, based on your experience of using Picard any help you can give to improve this documentation further will be most appreciated. Even if you cannot improve the existing help, if you can create or maintain translations into other languages, that would be of great benefit.

If you notice an error in the documentation or have additional material to contribute, please create a [ticket](#) under the Picard project (Documentation component). [Pull Requests](#) to address outstanding issues are also appreciated.

See also:

[Contributing to MusicBrainz Picard / Picard Translations / Contributing to the Documentation](#)

ACKNOWLEDGEMENTS

We gratefully acknowledge the following for their contributions to help develop, maintain and improve the Picard User Guide.

This list includes contributors to the documentation, regardless of their role. In addition to actual content contributors, this includes leads, translators, reviewers and proofreaders. All contributions are valuable and appreciated. As such, the names are presented in alphabetical order so as not to imply the relative importance of someone's contribution based on their position in the list.

Contributors include (in alphabetic surname order):

- Aerazol
- Vedant Chakravadhanula
- Pavan Chander
- Ronan Desplanques
- Gabriel Ferreira
- Giorgio Fontanive
- Rahul Kumar Gupta
- Wieland Hoffmann
- InvisibleMan78
- Jesus2099
- David Kellner
- Jun Kim
- Sambhav Kothari
- Soham Kukreti
- Laurent Monin
- Akash Nagaraj
- Frederik "Freso" S. Olesen
- Guntbert Reiter

- Theodore Fabian Rudy
- Suryansh Shakya
- skelly37
- Sophist
- Odd Stråbø
- Bob Swift
- Akshat Tiwari
- Philipp Wolfer
- yvanzo
- Shadab Zafar

Note: There are likely others that have not yet been identified, so we apologize if your name has been missed. Please let us know and we'll make sure that it is added to the list.

GLOSSARY OF TERMS

Many of the terms used in this documentation and within Picard itself have specific meaning in the MusicBrainz 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](#) for a full explanation of acoustic fingerprinting.

AcoustID

AcoustID is an acoustic fingerprint system built entirely on open-source technology. See the [AcoustID website](#) 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](#)”, while the albumartist for “[No Boundaries: A Benefit for the Kosovar Refugees](#)” is “Various Artists”.

Note: The albumartist usage is different for Classical Music releases, which follow the MusicBrainz [Classical Style Guide](#), 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 \(One in a Million\)](#)” on the album “[Tourist](#)”.

Please see the [Artist](#) page on the MusicBrainz website for additional information.

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

An artist credit indicates 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](#)” the artist is “[Walk off the Earth](#)” but is credited as “Gianni and Sarah”.

Please see the [Artist Credits](#) page on the MusicBrainz website for additional information.

caa

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

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

disc id

A Disc ID is the code number which MusicBrainz uses to link a physical CD to a release listing. It is a string of letters, like XzPS7vW.HPHsYemQh0HBUGr8vuU-. Disc IDs for a release can be seen on the disc IDs tab for the release on MusicBrainz. Clicking on these will give a detailed display of the disc ID, including the list of attached releases.

A release may have any number of disc IDs, and a disc ID may be linked to multiple releases. This is because [disc ID calculation](#) involves a hash of the frame offsets of the CD tracks. Different pressing of a CD often have slightly different frame offsets, and hence different disc IDs.

Conversely, two different CDs may happen to have exactly the same set of frame offsets and hence the same disc ID. For example [1wHl8fGzJyLXQR33ug60E8jh4k-](#) applies to a wide [variety of releases](#) by a variety of artists.

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](#) for more information.

medium

One of the physical, separate things you would get when you buy something in a record store. They are the individual CDs, vinyls, etc. contained within the packaging of an album (or any other type of release). Mediums are always included in a release, and have a position in said release (e.g. disc 1

or disc 2). They have a format like CD, 12" vinyl or cassette (in some cases this will be unknown), and can have an optional title (e.g. disc 2: The Early Years). For example, CD 1 of "[The Wall](#)".

Please see the [Medium](#) page on the MusicBrainz website for additional 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. Each track must always be associated with a single recording, but a recording can be linked to any number of tracks. For example, this recording of "[Bohemian Rhapsody](#)" is found as a track on over 100 releases.

Please see the [Recording](#) page on the MusicBrainz website for additional information.

release

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](#)" is one version of the album released by Great Big Sea.

Please see the [Release](#) 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 the "[Sea of No Cares](#)" release group contains multiple releases.

Please see the [Release Group](#) 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 \(live\)](#)" by Great Big Sea.

Please see the [Standalone Recording](#) 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](#)” by Jason Anderson is “[Driving Home](#)”.

Please see the [Track](#) 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. A work could also be a novel, play, poem or essay, later recorded as an oratory or audiobook. For example, the song “[Blinded by the Light](#)” written by Bruce Springsteen has been recorded well over 100 times.

Please see the [Work](#) page on the MusicBrainz website for additional information.

See also:

For more information on these and other terms used, please refer to the [Terminology](#) page on the MusicBrainz 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](#) webpage.

GETTING STARTED

This section provides information on how to get started using MusicBrainz Picard, including installation and some basic information about the user interface.

5.1 Download & Install Picard

MusicBrainz Picard is available for all major desktop operating systems (e.g. Windows, Linux and macOS), and in multiple forms (directly downloadable formal release executables, package manager versions of these, daily build executables, Python source code that you can execute with your own Python environment, etc.)

It is expected that most users will run formal release executables or package manager equivalents as these are easy to install, and are stable versions which are less likely to have bugs in experimental or new functionality.

However, any users wishing to contribute to the development of Picard or its Plugins may want to run from source code, downloading it from GitHub using a version of Git on their own computer. If you want to contribute to the Picard code but you don't understand what the previous sentence said, then you have a bit of a learning curve. :-)

The latest version of MusicBrainz Picard is always available for download from the [Picard Website](#). This includes installers for all supported platforms as well as [release source code](#). The very latest source code is also available at the [GitHub repository](#).

5.1.1 Installing Picard on Linux

Installing with Flatpak

Picard is available on [Flathub](#). This version should work on all modern Linux distributions, as long as Flatpak is installed (see [Flatpak Quick Setup](#)).

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
```

Installing with Snap

Picard is available as a Snap from the Snap Store. This version should work on all modern Linux distributions, as long as Snap is installed (see [Installing Snap](#)).

The [Snap Store page of Picard](#) gives detailed instructions on how to install Picard on various Linux distributions. If your Linux distribution supports it you can install Picard from your distribution's software center, e.g. Ubuntu Software or KDE Discover. You can also install Picard from the terminal:

```
snap install picard
```

Note: Picard installed as a Snap is running inside a sandbox and thus it does not have full access to all files and folders on your system. By default Picard has access to your home folder. You can additionally give it access to removable media by running the following command on a terminal:

```
snap connect picard:removable-media
```

Installing from your distribution's package repository

Picard is available in the package repositories of most distributions. The [download page](#) provides links to the packages for common Linux distributions. Please refer to your distribution's documentation for how to install software packages.

Please note that most distributions usually ship older versions of Picard. If you want to use the latest available version, as is recommended, install Picard as Flatpak or Snap as described above.

5.2 Starting Picard

Once Picard has been installed on your system, most of the time you will be starting it by clicking an icon on your desktop or in a start menu. This will run the program using the default location for the configuration file and configured logging level. Picard can also be started from a command line prompt with some overrides available. From the command line, you can also specify files or directories to load into Picard for processing. Please see [Appendix C: Command Line Options](#) for more details about the available options.

As of version 2.9, Picard will try to only run a single instance of the program at a time. When the program is started, it checks to see if there is another instance of that version, configuration file and plugin startup status `-P` already running. If the same version is already running, any files or directories specified on the command line of the new instance, along with any executable commands specified with the `-e` or `--exec` options will be passed to the already running instance for processing and the new duplicate instance will be shut down. This allows batch processing of files to be initiated automatically from other processes. If there is no instance of that version already running, Picard will start normally.

Additionally, Picard can be started in “stand-alone” mode, in which case it neither sends information to an already running instance nor accepts information from another instance.

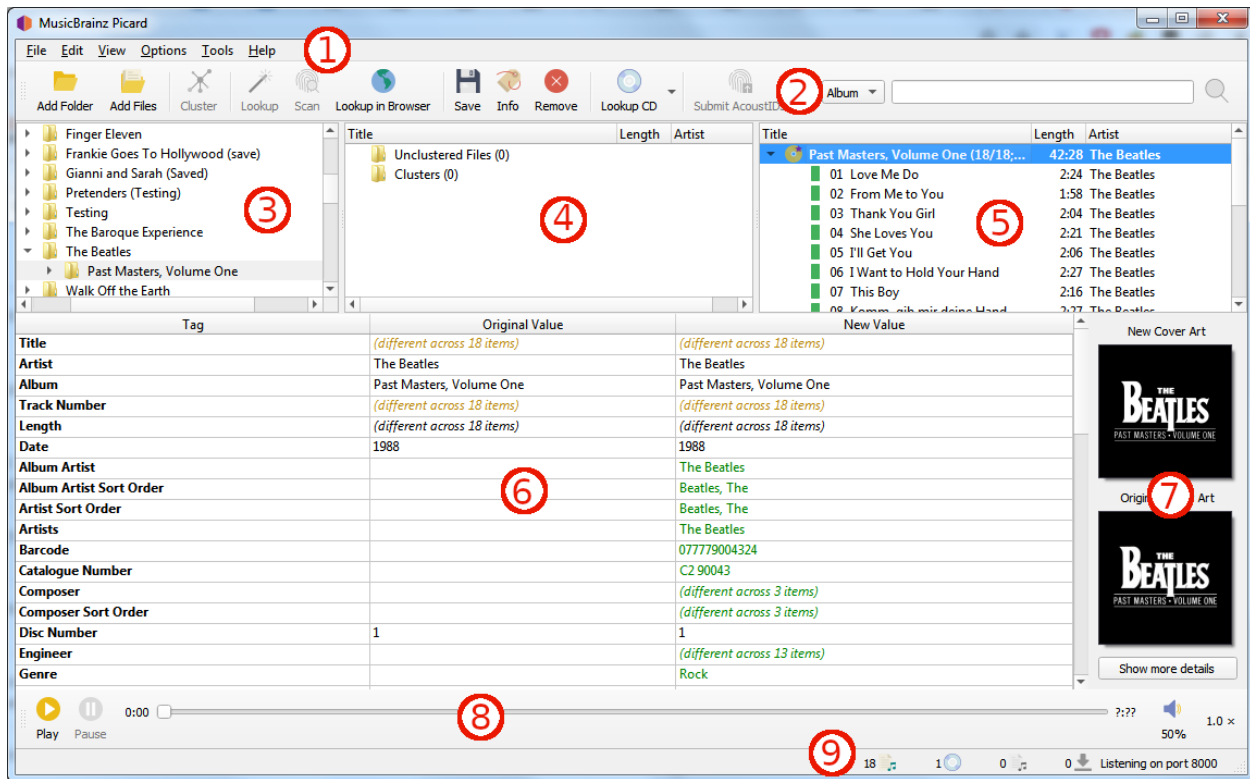
Instances started with command-line argument `-s / --stand-alone` always start as stand-alone.

If there is already an instance running when another instance is started that doesn't result in a stand-alone instance, any of the command-line overrides `-d / --debug`, `-M / --no-player` or `-N / --no-restore` of the new duplicate instance will be ignored, and only the specified files or directories and executable commands (if any) will be passed to the running instance for processing. Similarly, if a primary instance has been started with any of these overrides specified on the command line, starting a subsequent instance of that version without the override will not modify the user interface settings of the currently running instance.

All instances started with the `-h / --help`, `-v / --version` or `-V / --long-version` command line arguments will always output the requested product information and exit, regardless of whether or not another instance is running.

Please refer to the [Command and Batch Processing](#) section for more information.

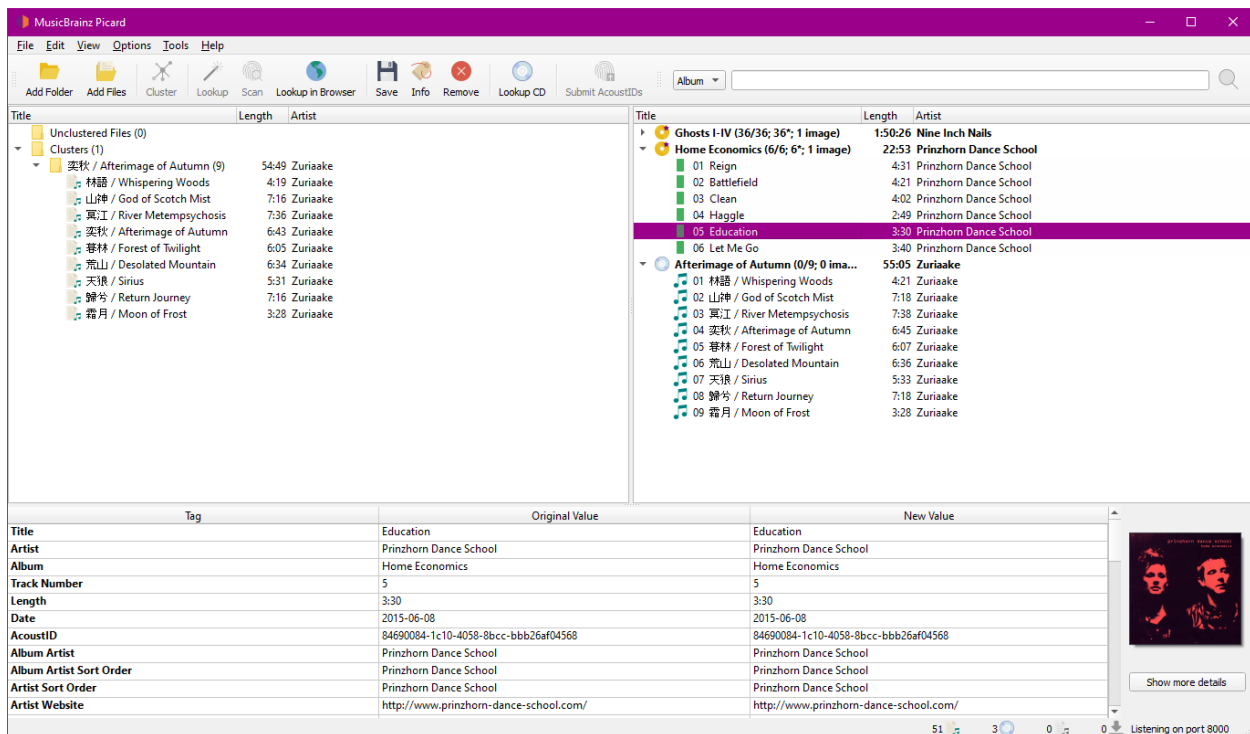
5.3 Main Screen



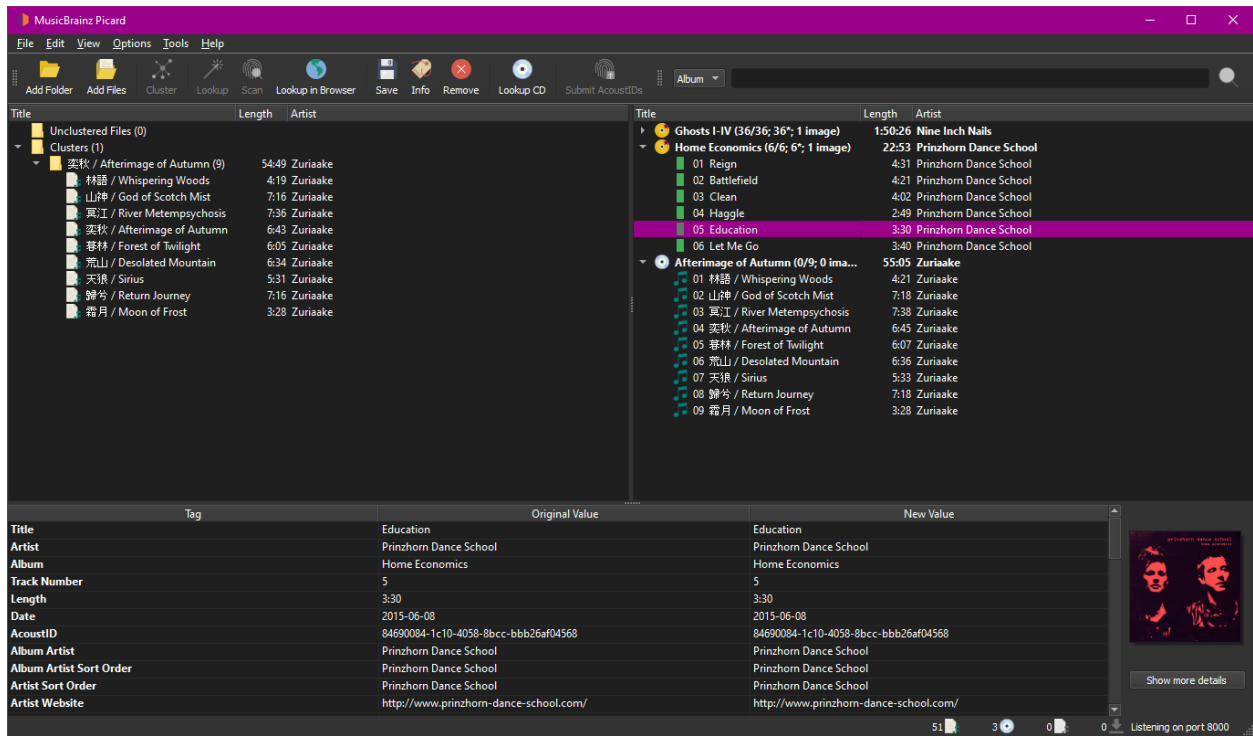
- 1. Menu Bar:** This provides the pull-down menu of actions that Picard can perform.
- 2. Tool Bar:** This provides quick links to the main functions performed by Picard. This can be customized by the user in the [User Interface Options](#) settings.
- 3. File Browser:** This provides a browser for 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.

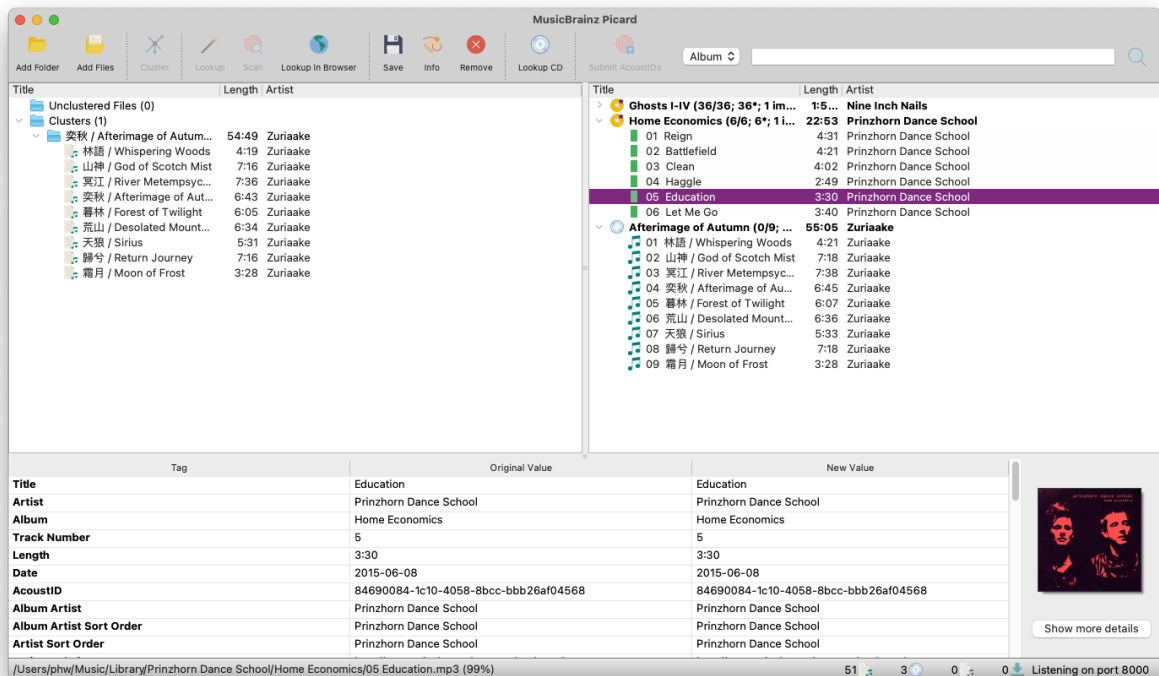
Picard is available for different operating systems. And while Picard's functionality is overall the same the specific look of the user interface can be slightly different based on the operating system. Throughout the documentation screenshots taken on different operating systems are used. Below you find a selection of screenshots of Picard's main screen on different systems:



Picard on Windows 10 (with light user interface)

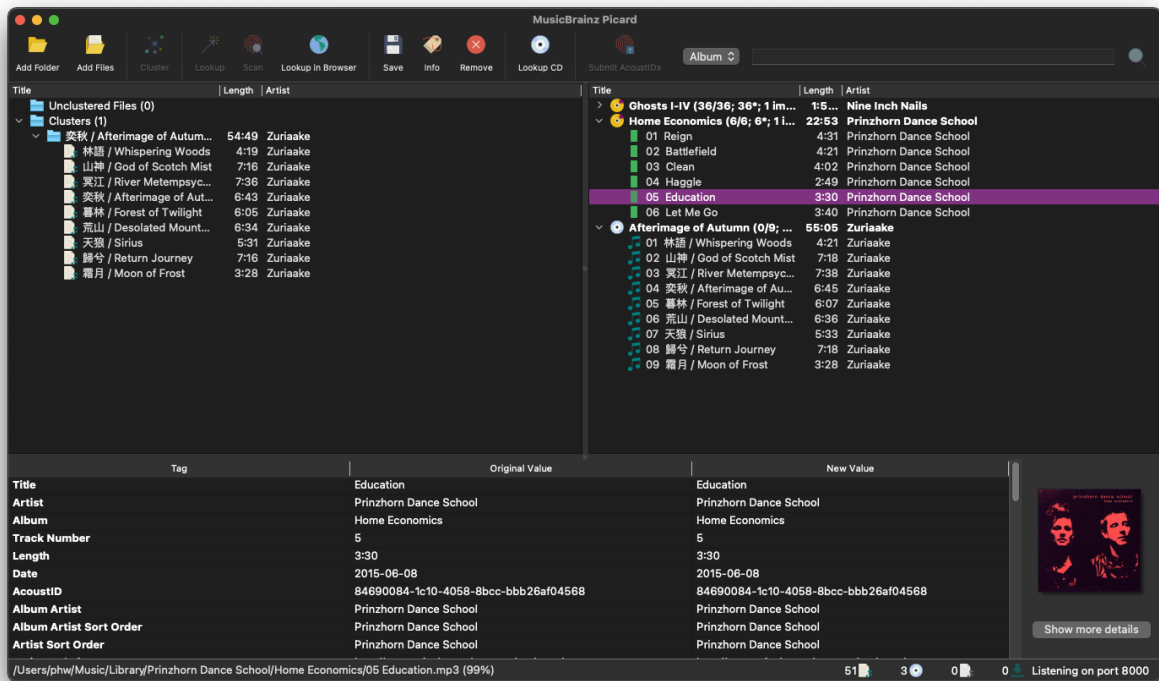


Picard on Windows 10 (with dark user interface)

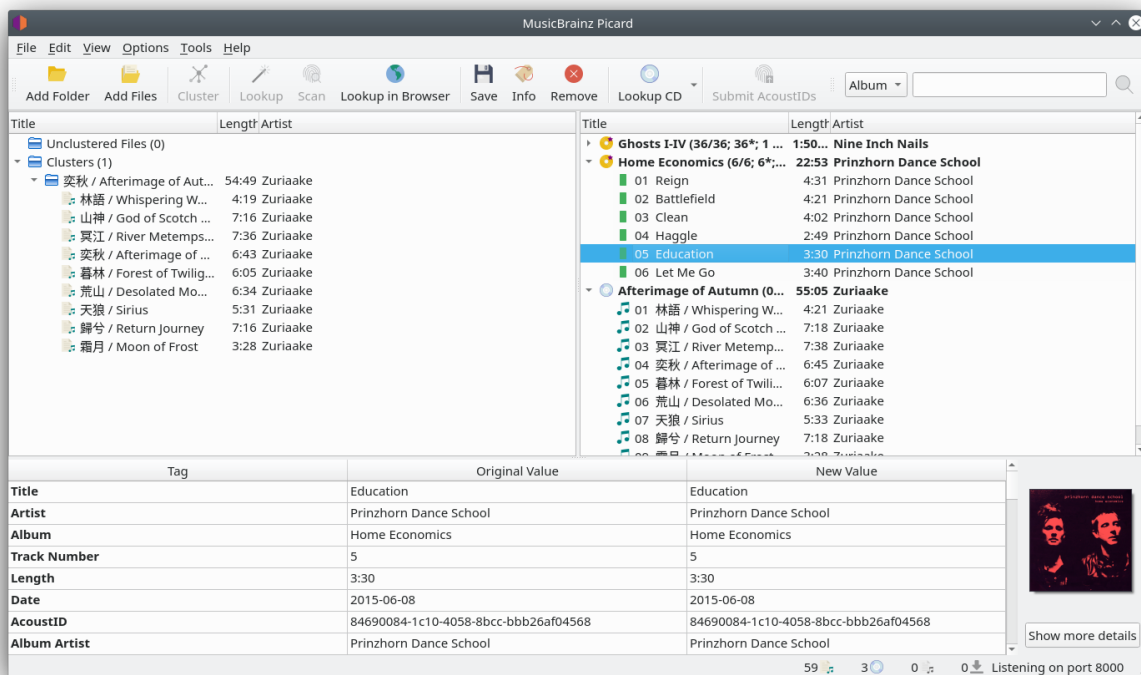


Picard on macOS

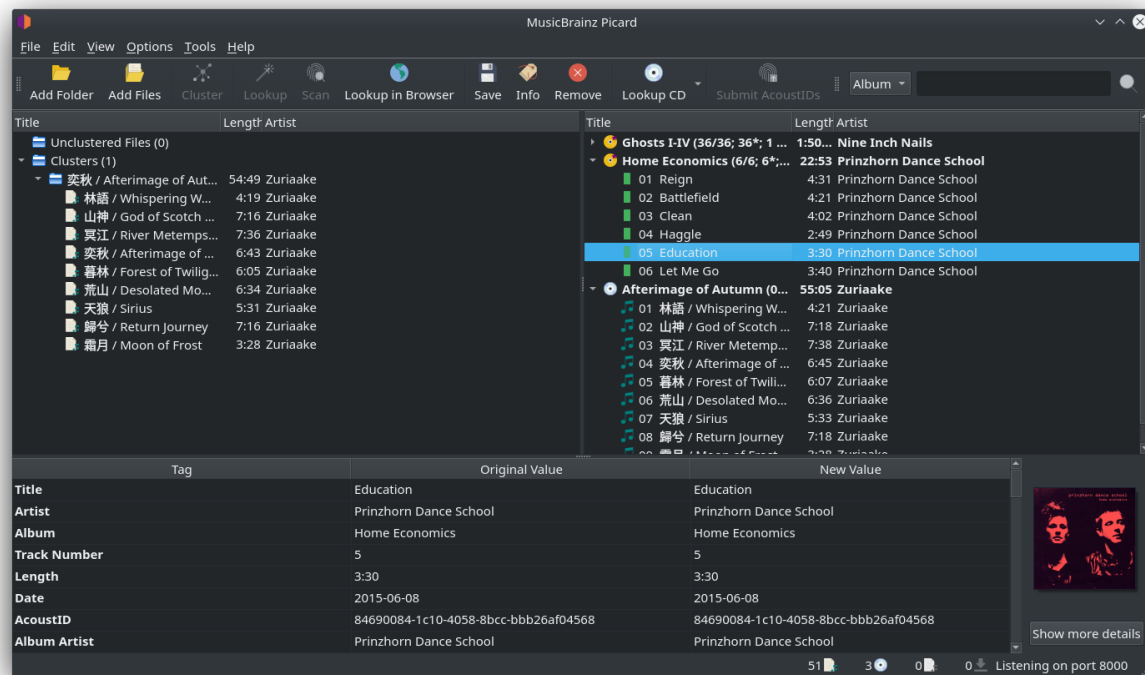
MusicBrainz Picard, Release v2.12.3



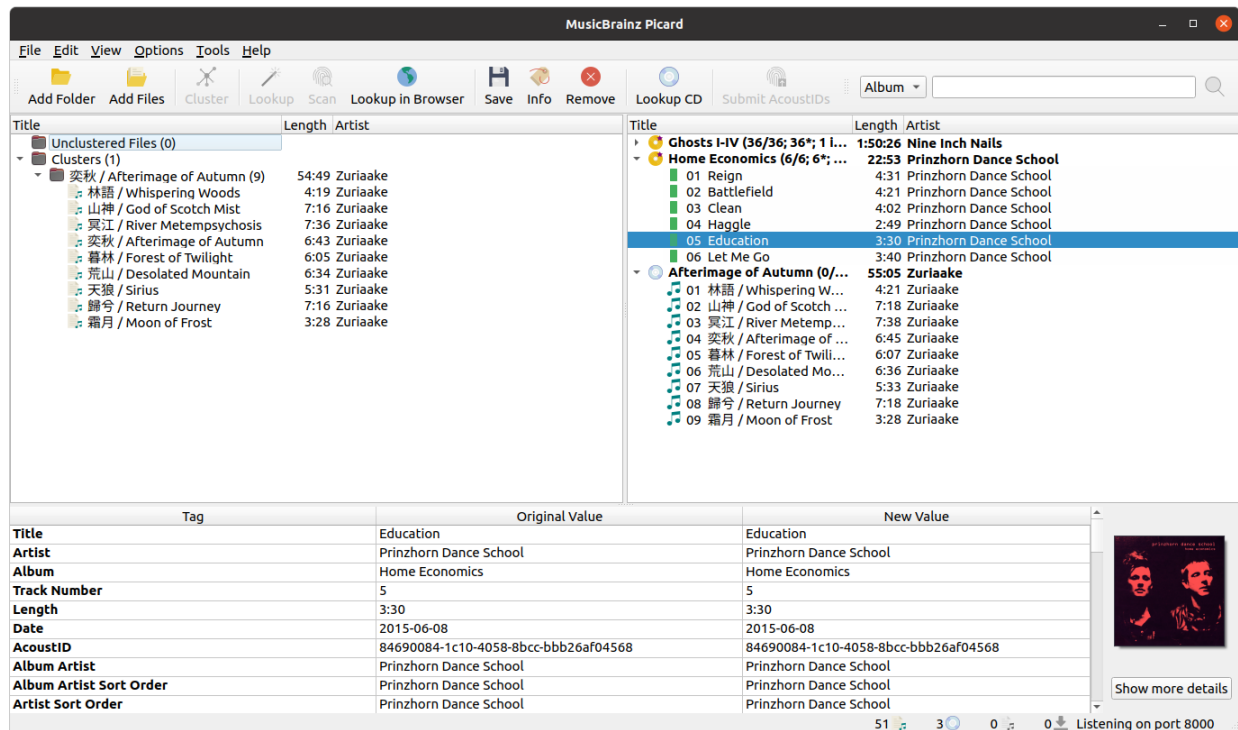
Picard on macOS (dark mode)



Picard on Linux with the KDE Plasma desktop environment (light theme)

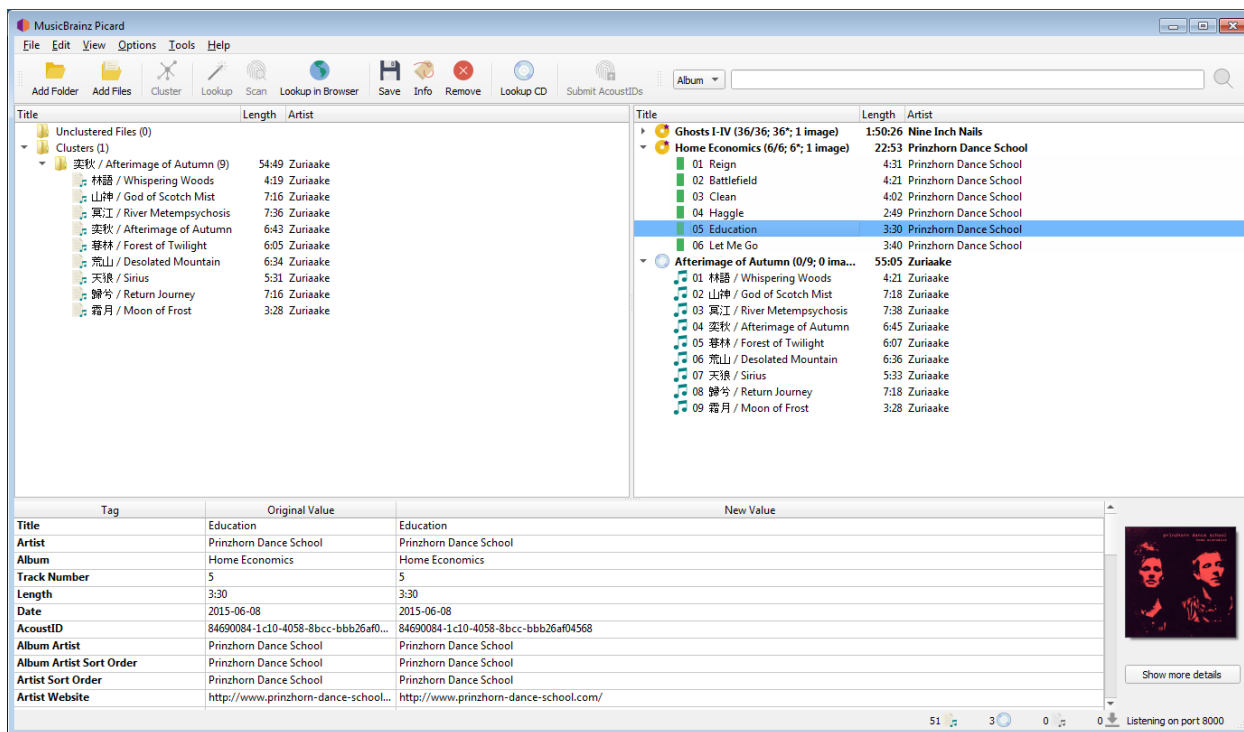


Picard on Linux with the KDE Plasma desktop environment (dark theme)

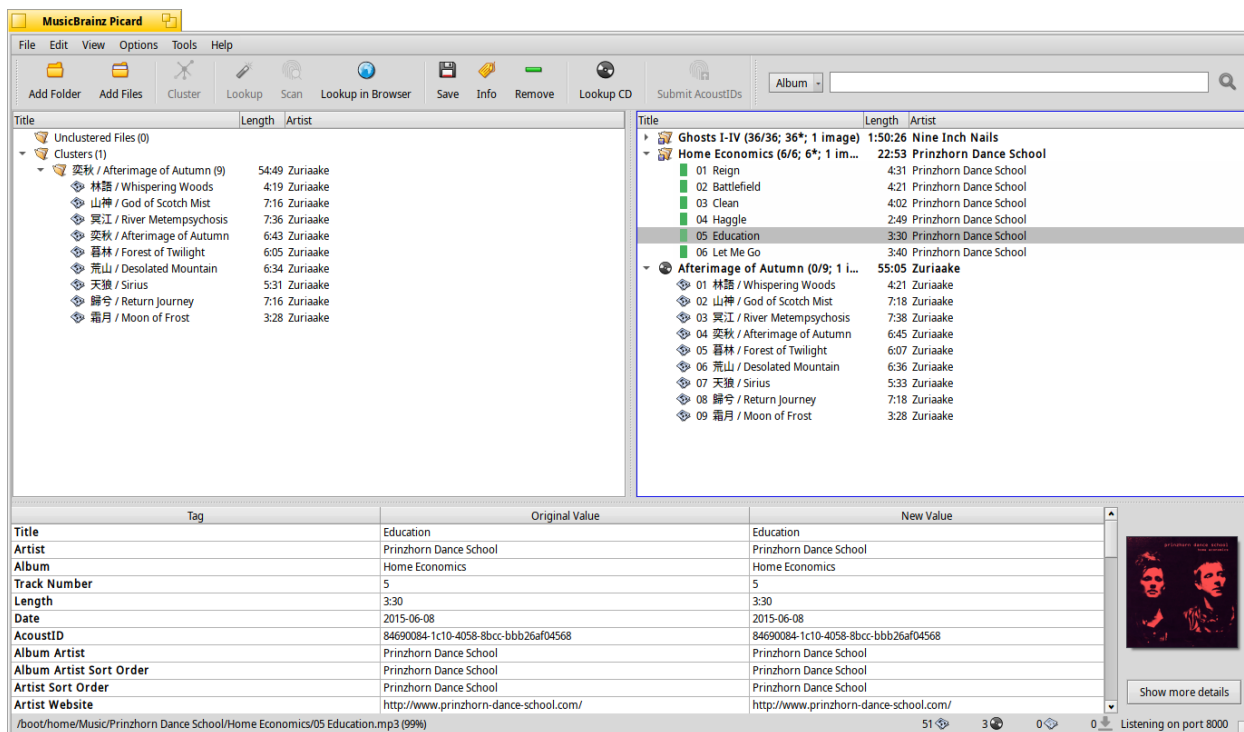


Picard on Linux with the GNOME desktop environment

MusicBrainz Picard, Release v2.12.3



Picard on Windows 7



Picard on Haiku

5.4 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.

5.4.1 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.

5.4.2 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, or 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, or 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, or if there are multiple files matched.



These icons indicate 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 a permission error while trying to load or save the file. This is typically due to the file being marked as read-only, or you do not have sufficient permission to read the file or save the file in the specified directory.



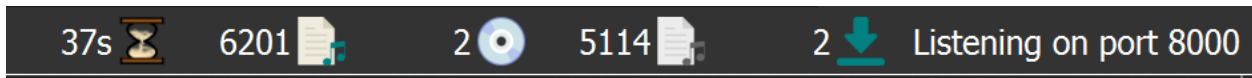
This icon indicates that Picard could not find the file with the given path while trying to load or save the track. This is typically due to the file being moved or deleted since it was loaded into Picard, but could also be due to a missing directory.



This icon indicates that Picard encountered an error while trying to load or save the track. This is typically due to a problem writing the tags into the file, but could also be due to a storage IO error on your system.

5.4.3 Status Bar

There is a status bar at the bottom of Picard's main screen, which displays some information about current processing status. This includes four numbers along with the current port number that Picard is monitoring. In addition, if Picard is currently processing your files an estimated time to completion will also be displayed to the left of the status indicators.



From left to right, the numbers represent:

1. The estimated processing time remaining (only displayed if Picard is actively processing files).
2. The number of files loaded.
3. The number of MusicBrainz releases loaded.
4. The number of files with pending action (e.g. loading, saving, fingerprinting).
5. The number of active network requests.

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.

6.1 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 can be loaded into Picard by dragging and dropping them to the right panes, double clicking on individual files or by selecting multiple files and folders and selecting “Load selected files” from the context menu.

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.

6.2 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.

6.3 Option Settings

The option settings are found under the “*Options* → *Options...*” item on the menu bar. On macOS they can be accessed with “*MusicBrainz Picard* → *Preferences...*”. 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.

Note: When running your code from the source in a macOS environment, you can access the option settings by navigating to the “*Python* → *Preferences...*” option in the menu bar. This allows you to configure and customize various settings for your development environment.

In addition to the basic “user settings”, this is also where option setting changes are made to individual option profiles. This is covered in greater detail in the [Option Profiles](#) section.

Changes made to a profile’s options settings, enabled status, or position in the profile stack will be reflected in the option settings displayed on the other pages. Options that are controlled by an enabled profile will be shown as highlighted. Hovering your cursor over the highlighted option will identify which profile currently controls the setting. Settings are always displayed based on the first enabled profile in the profile stack, which corresponds to the setting that will be used during processing.

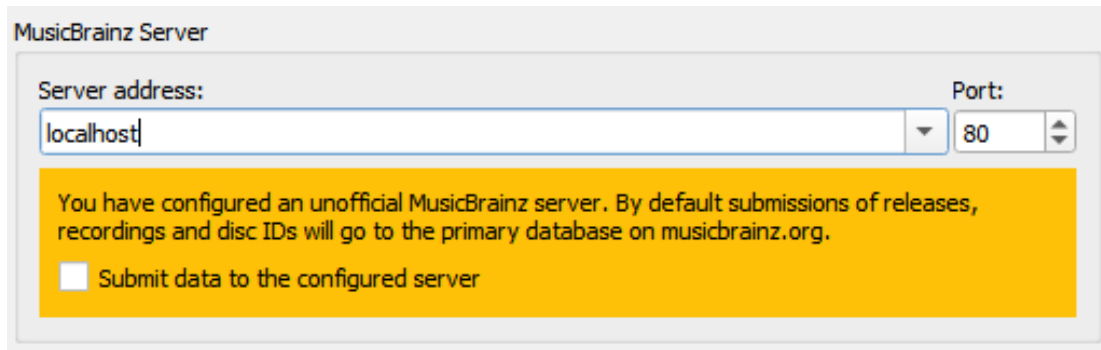
6.3.1 General Options

The screenshot shows the 'MusicBrainz Server' section with a 'Server address' field containing 'musicbrainz.org' and a 'Port' spinner set to '443'. Below this is the 'MusicBrainz Account' section with a 'Log in' button. The 'General' section contains three unchecked checkboxes: 'Automatically scan all new files', 'Automatically cluster all new files', and 'Ignore MBIDs when loading new files'. The 'Update Checking' section has two checked checkboxes: 'Check for plugin updates during startup' and 'Check for program updates during start-up'. It also features a 'Days between checks' spinner set to '7' and an 'Updates to check' dropdown menu currently showing 'Stable releases only'.

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).

In addition to the standard MusicBrainz servers provided in the drop down list, you can manually enter an alternate address, such as “localhost” if you are running a local copy of the server. When an alternate server host name is entered, a warning will be displayed and you will be asked to confirm that you want to submit all data to this alternate server.



Port

The port number for the server. Default value: 443 (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"*. See also *Scan Files* and [Understanding Acoustic Fingerprinting and AcoustIDs](#).

Automatically cluster all new files

Check this box if you want Picard to automatically group all loaded files into album clusters. Leave it unchecked if you don't want Picard to do this automatically. In any case, you can direct Picard to cluster files any time using *"Tools → Cluster"*. See also *Lookup Files*.

Note: You can either enable "Automatically scan all new files" or "Automatically cluster all new files", but not both.

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 plugin updates during start-up

This option determines whether or not Picard will automatically check for plugin updates during startup. If this is enabled and an update to an installed plugin is available, a popup message will be displayed.

Check for program 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 program 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 program updates during start-up” option is enabled.

Updates to check

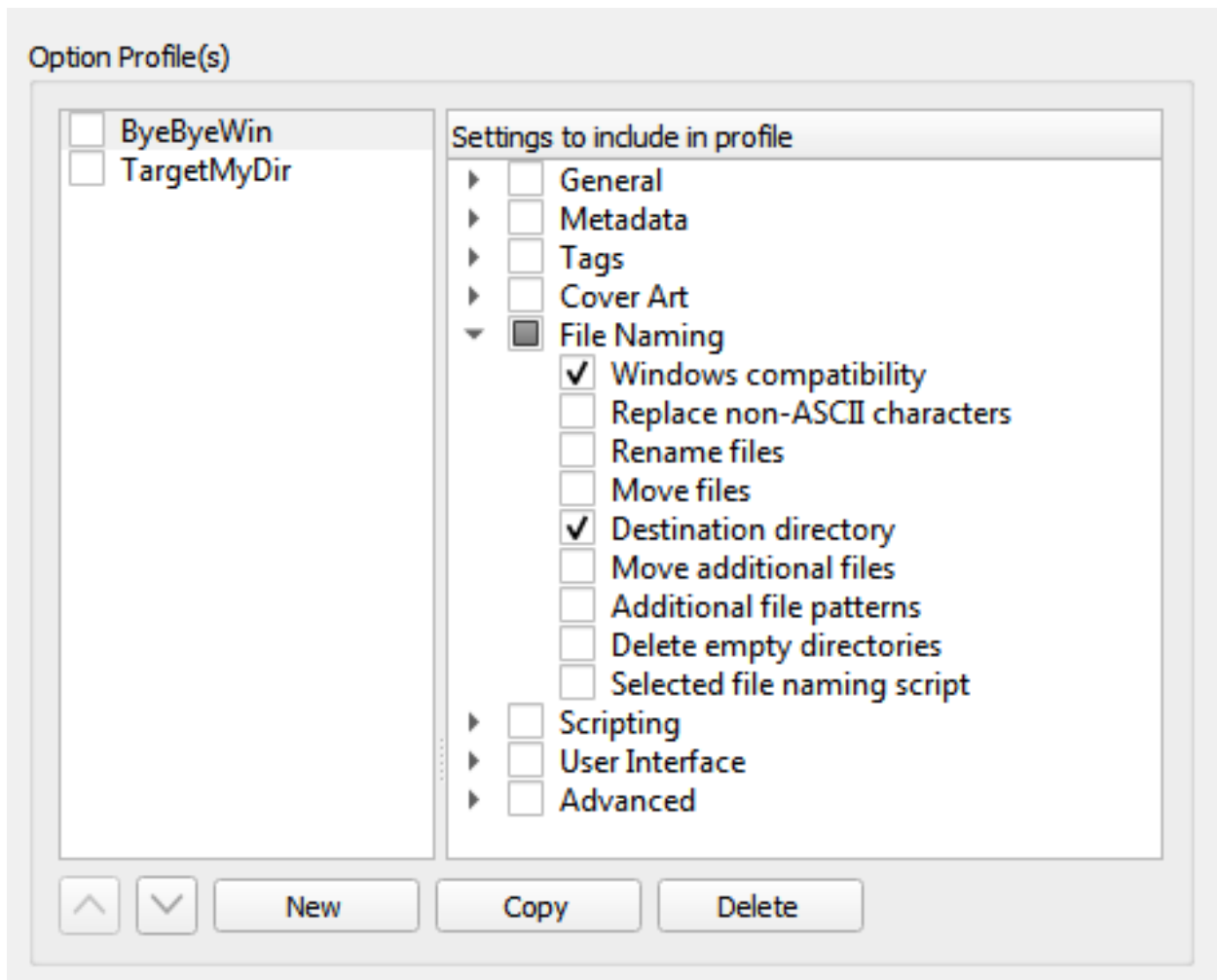
This option allows you to select which levels of program update to check. Your options are:

- 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 program 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.

6.3.2 Profile Options



As of version 2.7, Picard supports multiple profiles that can quickly switch between option settings. This page allows for the management of those user-defined option profiles.

Initially, the list of profiles will be empty. To create a new profile click on the *New* button. This will create a profile with no options selected for the profile to manage. To rename the profile, right-click on the profile name and select the “*Rename profile*” command.

The options that the profile is to manage are selected from the list in the right-hand pane. Options can be selected either by group or individually. The groups can be expanded to see the individual options belonging to that group.

The profile stack order can be rearranged either by selecting a profile and using the up and down arrow buttons below the list, or by dragging the profile to a new position in the stack. Profiles are enabled when the box beside the profile’s name is checked.

Changes made to a profile’s options settings, enabled status, or position in the profile stack will be reflected in the option settings displayed on the other pages. Options that

are controlled by an enabled profile will be shown as highlighted. Hovering your cursor over the highlighted option will identify which profile currently controls the setting. Settings are always displayed based on the first enabled profile in the profile stack, which corresponds to the setting that will be used during processing.

Warning: It is important to understand that when you click the *Make It So!* button **all** of the option settings on **all** pages will be saved. If an option is managed by one or more profiles that are currently enabled, the option will be highlighted and it will be saved to the **first** enabled profile in the profile stack that manages the option. If there are no enabled profiles that manage the option, the option will not be highlighted and it will be saved to the “user settings” profile which is the user’s normal settings, contains all options, is at the bottom of the profile stack, and is always enabled. The “user settings” profile cannot be modified and is not shown in the profile management page.

See also:

Please see the [Option Profiles](#) section for a detailed explanation of the profile system.

6.3.3 Metadata Options

The screenshot shows the 'Metadata' section of the MusicBrainz Picard settings. It contains several checkboxes and text input fields with 'Select...' buttons.

Metadata

- ☒ Translate artist names to these locales where possible:
 - English Select...
- ☐ Ignore artist name translation for these scripts:
 - Select...
- ☐ Use standardized artist names
- ☒ Use standardized instrument and vocal credits
- ☒ Convert Unicode punctuation characters to ASCII
- ☒ Use release relationships
- ☒ Use track relationships
- ☒ Guess track number and title from filename if empty

Custom Fields

Various artists:

- Various Artists Default

Standalone recordings:

- [standalone recordings] Default

Translate artist names to this locale where possible

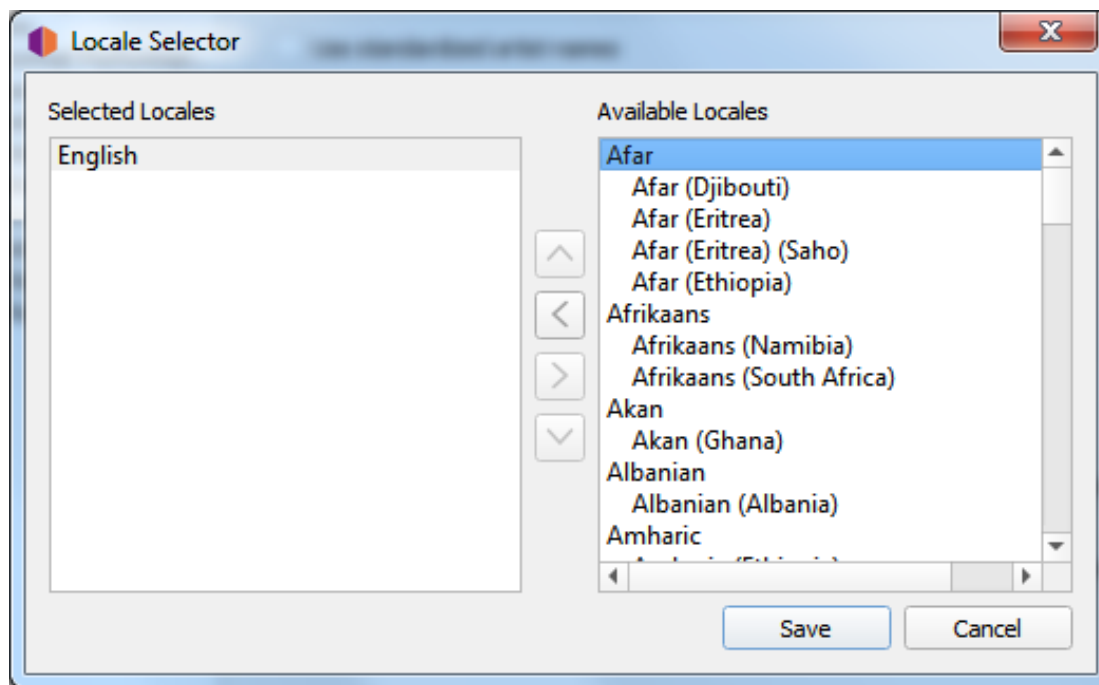
When checked, Picard will check each of the selected locales in order to see whether an artist has an alias for that locale. If it does, Picard will use that alias instead of the artist name when tagging. For example, if you have selected locales of “English (Canadian)” and “English (US)”, and there are aliases for “English (US)”, “English” and “Greek”, then the “English (US)” alias will be used.

Note that Picard will attempt to use the first exact match first. For example, if you have selected locales of “English (Canadian)”, “English (US)” and “Greek”, and there are aliases for “English” and “Greek”, then the “Greek” alias will be used.

If there are no exact matches to any of the selected locales, then Picard will attempt to find a match based on the root locale. For example, if you have selected locales of “English (Canadian)”, “English (US)” and “Greek (Cyprus)”, and there are aliases for “English (UK)” and “Greek”, then the “English (UK)” alias will be used.

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.

To select which locales to use, click the *Select...* button beside the list of selected locales. This will bring up a new dialog window where you can add, remove or reorder your list of selected locales.



Once you are satisfied with your selections, click the *Save* button to transfer the list to your option settings and close the dialog. Note that the changes will not be saved permanently until you click the *Make It So!* button.

Ignore artist name translation for script

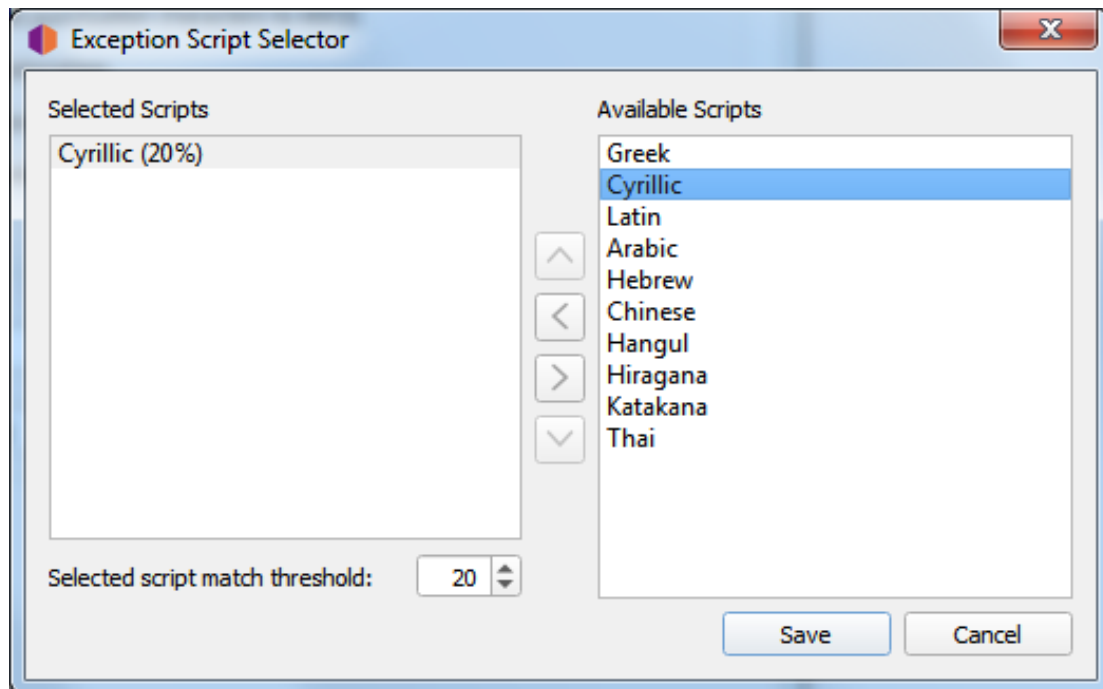
Sometimes you may not want to have the artist names translated if they appear in a certain character set script. When checked, this option will tell Picard to not perform the artist name translation if it is written using one of the selected scripts.

Each selected script includes a matching threshold value used to determine if that script should be used. When an artist name is evaluated to determine if it matches one of your selected scripts, it is first parsed to determine which scripts are represented in the name, and what weighted percentage of the name belongs to each script. Then each of your selected scripts are checked, and if the name contains characters belonging to the script and the percent-

age of script characters in the name meets or exceeds the match threshold specified for the script, then the artist name will not be translated.

For example, if “Translate artist names” is enabled with the locale set to “English”, and you enable “Ignore artist name translation” with the scripts set to “Greek (30%)” and “Cyrillic (50%)”, any artist names that contain 30% Greek characters or 50% Cyrillic characters will not be translated and will appear in their origin form.

To select which character set scripts to use and their weighting thresholds, click the *Select...* button beside the list of selected scripts. This will bring up a new dialog window where you can add, edit or remove items in your list of selected scripts.



Once you are satisfied with your selections, click the *Save* button to transfer the list to your option settings and close the dialog. Note that the changes will not be saved permanently until you click the *Make It So!* button.

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.

Guess track number and title from filename if empty

If checked, Picard will try to guess a file’s track number or title from the filename if the tracknumber or title tag is empty.

Various artists

Choose how you want the “Various Artists” artist spelled.

Standalone recordings

Choose how you want “Standalone recordings” to be grouped.

Preferred Releases

Preferred release types

Album	<input type="range"/>	Audiobook	<input type="range"/>	Mixtape/ Street	<input type="range"/>
Single	<input type="range"/>	Compilation	<input type="range"/>	Remix	<input type="range"/>
EP	<input type="range"/>	DJ-mix	<input type="range"/>	Soundtrack	<input type="range"/>
Other	<input type="range"/>	Demo	<input type="range"/>	Spokenword	<input type="range"/>
Broadcast	<input type="range"/>	Interview	<input type="range"/>	<input type="button" value="Reset all"/>	
Audio drama	<input type="range"/>	Live	<input type="range"/>		

Preferred Release Types

Adjust the sliders on the right-hand side of each of the various release types to tweak how likely Picard is to match a file or cluster to releases of that types. Moving a slider to the right increases the likelihood of matching that type, while moving the slider to the left decreases the likelihood.

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 Medium 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

☐ Only use my genres

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

☒ Use folksonomy tags as genre

Minimal genre usage:

Maximum number of genres:

Join multiple genres with:

Genres or folksonomy tags to include or exclude, one per line:

-seen live
-favorites
-fixme
-owned

Playground for genres or folksonomy tags filters (cleared on exit):

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**

Picard saves the ratings together with an e-mail address identifying the user who did the rating. That way different ratings for different users can be stored in the files. Please specify the e-mail you want to use to save your ratings.

E-mail:

users@musicbrainz.org

☒ **Submit ratings to MusicBrainz**

Enable track ratings

Check to write track ratings to your files.

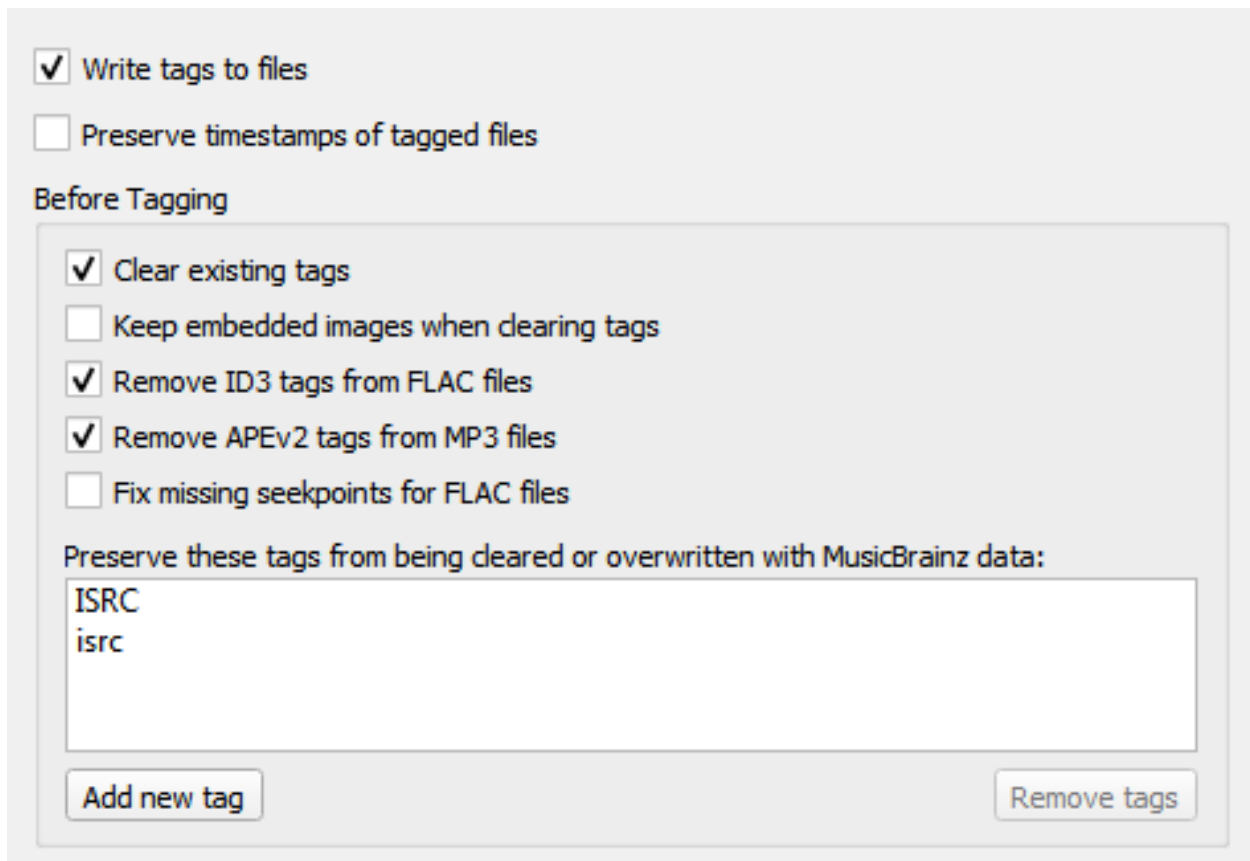
E-mail

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.

6.3.4 Tag Options



The screenshot shows the 'Tag Options' dialog box in MusicBrainz Picard. It has a light gray background. At the top, there are two checkboxes: 'Write tags to files' (checked) and 'Preserve timestamps of tagged files' (unchecked). Below these is a section titled 'Before Tagging' in a slightly darker gray. Inside this section, there are five checkboxes: 'Clear existing tags' (checked), 'Keep embedded images when clearing tags' (unchecked), 'Remove ID3 tags from FLAC files' (checked), 'Remove APEv2 tags from MP3 files' (checked), and 'Fix missing seekpoints for FLAC files' (unchecked). Below the checkboxes is a text box labeled 'Preserve these tags from being cleared or overwritten with MusicBrainz data:'. The text box contains the text 'ISRC' and 'isrc' on two lines. At the bottom of the dialog, there are two buttons: 'Add new tag' on the left and 'Remove tags' on the right.

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.

Keep embedded images when clearing tags

The default is for Picard to remove any embedded images from the files when clearing existing tags. Checking this option will keep the embedded images in the files.

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.

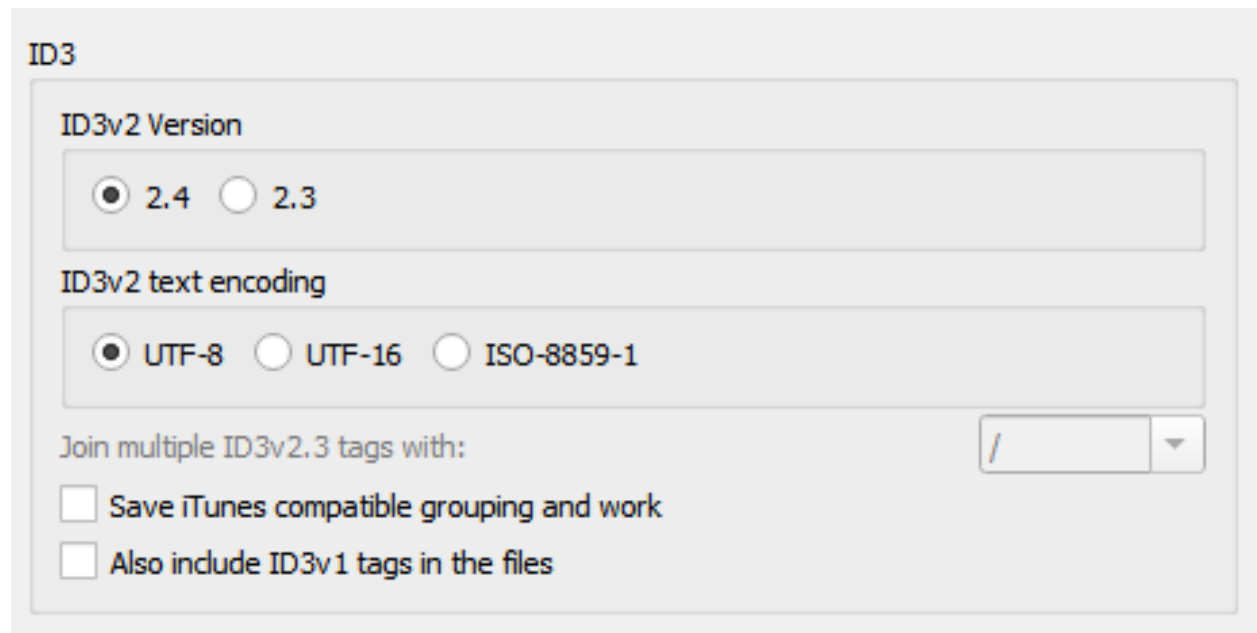
Fix missing seekpoints for FLAC files

Some software has issues handling FLAC files that have an empty seek table metadata block. When this option is enabled empty and hence unused seek table blocks will be removed from the files on saving.

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.

ID3 Files



The screenshot shows the 'ID3' configuration window. It has a title bar 'ID3'. Inside, there's a section 'ID3v2 Version' with two radio buttons: '2.4' (selected) and '2.3'. Below that is 'ID3v2 text encoding' with three radio buttons: 'UTF-8' (selected), 'UTF-16', and 'ISO-8859-1'. To the right of these is a field 'Join multiple ID3v2.3 tags with:' followed by a text box containing '/' and a dropdown arrow. At the bottom, there are two checkboxes: 'Save iTunes compatible grouping and work' (unchecked) and 'Also include ID3v1 tags in the files' (unchecked).

ID3v2 version

ID3v2.4 is the latest version and the default since Picard 2.9. Most modern software and devices can read ID3v2.4 tags. If you encounter issues with tag reading with your music player try using v2.3 instead.

Other than native support for multi-valued tags in v2.4, the [Picard Tag Mapping](#) section 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.

Note: This setting is explicitly only for ID3 v2.3 tags, because those don’t support proper multi value tags. Hence this separator setting is actually more of a workaround to this fact.

Because Vorbis tags allow the same tag to exist multiple times, and hence support multi valued tags by default, there actually is no separator at all there. It is just multiple values, shown using whatever the player software uses as a separator.

If for some reason you want all the values of the tag in a single field separated by some separator you could use a script like:

```
$set(musicbrainz_artistid,$join(%musicbrainz_artistid%, / ))
```

to flatten all the values into a single field separated by “/”.

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.

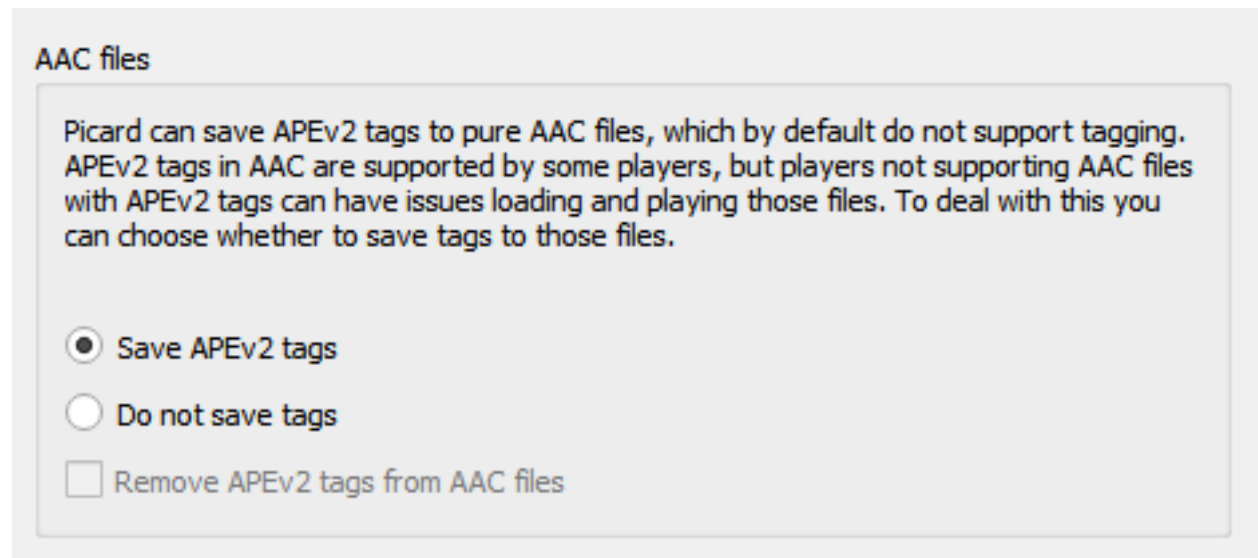
See the [Picard Tag Mapping](#) section for details.

Note: For other players supporting grouping and work you might need to disable this option. [MusicBee](#) 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 Files



Picard can save APEv2 tags to pure AAC files, which by default do not support tagging. APEv2 tags in AAC are supported by some players, but players not supporting AAC files with APEv2 tags can have issues loading and playing those files. 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

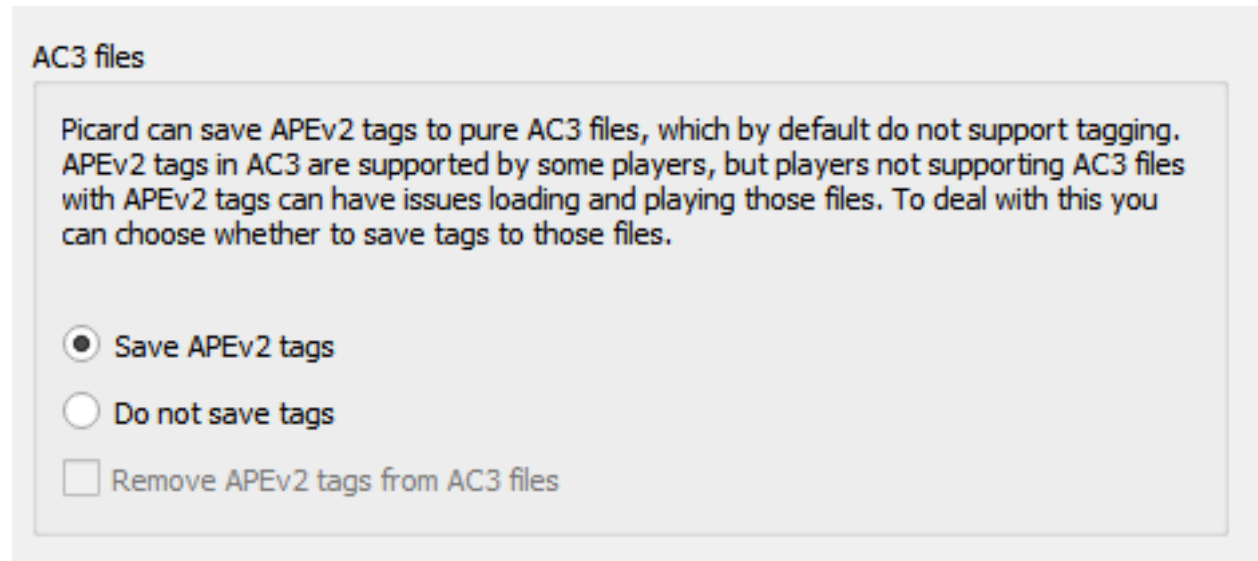
Picard will not save any tags to the files, but you can still use Picard to rename them. By default existing APEv2 tags will be kept in the file.

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 files.

AC3 Files



Picard can save APEv2 tags to pure AC3 files, which by default do not support tagging. APEv2 tags in AC3 are supported by some players, but players not supporting AC3 files with APEv2 tags can have issues loading and playing those files. 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 AC3 files.

WAVE Files



The screenshot shows a configuration window titled "WAVE files". It contains a text box explaining that Picard tags WAVE files with ID3v2 tags, but some software may not support them. It offers an option to write RIFF INFO tags instead, which have limited support. Below this, there are two checkboxes: "Also include RIFF INFO tags in the files" (checked) and "Remove existing RIFF INFO tags from WAVE files" (unchecked). At the bottom, there is a section for "RIFF INFO Text Encoding" with two radio buttons: "Windows-1252" (selected) and "UTF-8".

WAVE files

Picard will tag WAVE files using ID3v2 tags. This is not supported by all software. For compatibility with software which does not support ID3v2 tags in WAVE files additional RIFF INFO tags can be written to the files. RIFF INFO has only limited support for tags and character encodings.

☒ Also include RIFF INFO tags in the files

☐ Remove existing RIFF INFO tags from WAVE files

RIFF INFO Text Encoding

☒ Windows-1252 ☐ UTF-8

WAVE by itself as a standard only supports the INFO chunk tags, which are very limited. In addition, INFO chunks don't have any proper support for encoding.

In all cases Picard will write ID3 tags to the WAVE files. This is supported by quite a few tools; however, it is not supported universally. Tools not supporting ID3 tags should just ignore them. If possible, this is likely the best option to have proper tags in WAVE files. For compatibility with software that does not support ID3v2 tags, Picard can also save [Resource Interchange File Format](#) (RIFF) INFO tags to WAVE files. RIFF INFO is read and written only as an extra. If there are no existing ID3 tags, the data from RIFF INFO will be used. When saving files, RIFF INFO will be written in addition to the ID3v2 tags.

Picard's use of the RIFF INFO tags is determined by the following configuration settings:

Also include RIFF INFO tags in the files

Picard will save the RIFF INFO tags to the files.

Remove existing RIFF INFO tags from WAVE files

Picard will remove any existing RIFF INFO tags from the WAVE files. This setting is ignored if the previous setting is enabled to allow writing the RIFF INFO tags to the files.

RIFF INFO Text Encoding

This setting allows you to specify the encoding used for the RIFF INFO tags written. The default setting is Windows-1252 encoding. Picard can also use UTF-8 as an alternative, which allows for full language support, but it depends on the software reading it. Typically, if the software supports this, it will read the ID3 tags anyway so there is not much to be gained.

6.3.5 Cover Art Options

The screenshot shows the 'Cover Art Options' dialog box in MusicBrainz Picard. It has a light gray background and contains several sections of settings.

- Embed cover images into tags:** A checked checkbox.
- Embed only a single front image:** A checked checkbox inside a rounded rectangle.
- Save cover images as separate files:** A checked checkbox.
- Use the following file name for images:** A text input field containing the word 'cover'.
- Overwrite the file if it already exists:** An unchecked checkbox.
- Save only a single front image as separate file:** A checked checkbox.
- Always use the primary image type as the file name for non-front images:** An unchecked checkbox.
- Cover Art Providers:** A section with a list of providers, each with a checkbox:
 - ☒ fanart.tv
 - ☐ Cover Art Archive: Release
 - ☐ Allowed Cover Art URLs
 - ☐ Cover Art Archive: Release Group
 - ☐ Amazon
 - ☒ Local Files
- Reorder Priority:** Two buttons with up and down arrows.

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.

Embed only a single front image

Embeds only a single front image into your music files. No other images, regardless of their type, will be embedded. 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 display the image as a preview of 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.

For example, specifying a file naming mask such as:

```
%albumartist% - %originalyear% - %album% - %coverart_maintype%
```

will preface the file name with the album artist, original release year and album title.

You can also have Picard save the images to a subdirectory by including this in the file naming mask. For example:

```
Artwork/%albumartist% - %originalyear% - %album% - %coverart_
↪ maintype%
```

which will place the images in a subdirectory called "Artwork".

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.

Save only a single front image as separate file

This tells Picard to only save the first “front” image to a separate file with the release. No other “front” images or images of any other type will be saved. If left unchecked, all “front” images will be saved as separate files, along with any other specified image types to be downloaded.

Always use the primary image type as the file name for non-front images

This setting changes how Picard names image files **other than front images**.

When checked, Picard will use the type of the image (e.g.: back, booklet, etc.) as the filename when saving, as long as the type is not front. If the image has been assigned multiple types, then the first type will be used. For example, if the image is of types “back” and “raw”, then “back” will be used for the filename. If unchecked or if the image is of type “front”, Picard will use the file name specified in the “Use the following file name for images” setting.

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: Release

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

Cover Art Archive: 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 than having an exact representation of your release. It is also a good fallback for the Cover Art Archive provider.

Allowed Cover Art URLs

This will use images provided from approved third-party sites. The image location is stored as a URL relationship for the release within the MusicBrainz database, and only approved sites can be used for this relationship.

Note: This relationship type is now deprecated in MusicBrainz, and is no longer used. See [Cover art whitelist](#) in the Style Guide for more information.

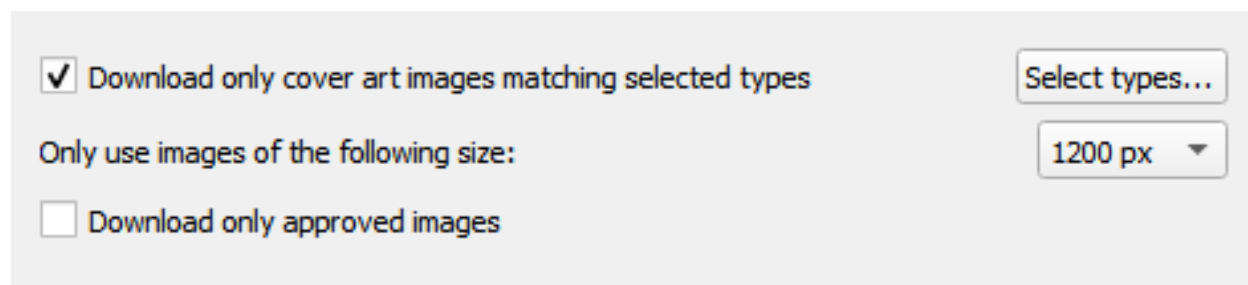
Local Files

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*.

- **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, 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](http://TheAudioDB.com), 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



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 only cover art images matching the selected 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.

Only use images of the following size

This identifies what size of image to download from the CAA. The options are 250px, 500px, 1200px and 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.

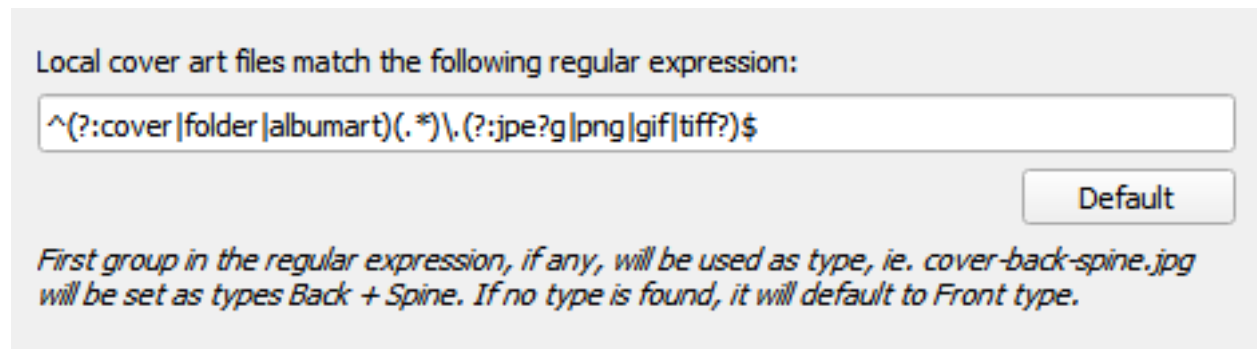
Note: If the selected size is not available, then Picard will use the largest available size below the selected size.

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.

Note: 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



Local cover art files match the following regular expression:

`^(?:cover|folder|albumart)(.*)\.(?:jpe?g|png|gif|tiff?)$`

Default

First group in the regular expression, if any, will be used as type, ie. cover-back-spine.jpg will be set as types Back + Spine. If no type is found, it will default to Front type.

In this section you can configure the file names to be used by the Local Files cover art provider. If you are trying to collect more than one image, the naming is important.

The file names are defined using a regular expression. The default is `^(?:cover|folder|albumart)(.*)\.(?:jpe?g|png|gif|tiff?)$` which will 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”).

The first part of the regular expression is a non-capture group: `(?:cover|folder|albumart)`. Items listed in this group will not get captured and the default (Front) type will apply.

The second part of the regular expression is a group: `(.*)`. This is the real capture, so if the file names match any of the cover art types, they will be tagged as such.

Note: A common mistake is to add all the types into the first (non-capture) group. This means that all the regular file names would be thrown into the Front type and cause unexpected results.

6.3.6 File Naming Options

☒ Move files when saving

Destination directory:

C:\CD Rip\Tagged\Picard Portable Browse...

☐ Move additional files (case insensitive):

*.jpg *.png

☒ Delete empty directories

☒ Rename files when saving

Selected file naming script:

User: Primary file naming script Edit file naming script...

Files will be named like this:

Before	After
track05.mp3	C:\CD Rip\Tagged\Picard Portable\
ticket_to_ride.mp3	C:\CD Rip\Tagged\Picard Portable\

If you select files from the Cluster pane or Album pane prior to opening the Options screen, up to 10 files will be randomly chosen from your selection as file naming examples. If you have not selected any files, then some default examples will be provided.

Reload examples

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: 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 directory 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 support the Unix shell-style wildcards, and are separated by spaces. The wildcard patterns available are:

Pattern	Meaning
*	matches everything
?	matches any single character
[seq]	matches any character in <i>seq</i>
[!seq]	matches any character not in <i>seq</i>

For a literal match, wrap the meta-characters in brackets. For example, ‘[?]’ matches the character ‘?’.

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”.

Note: This option can also be used to move subdirectories to the new release directory. This is done by specifying the name of the subdirectory

in the list of files to be moved. For example, if your album folders have a subfolder called “Artwork”, “covers” or “scans” that contains additional image files that you also want to move to the new release directory, simply add “artwork”, “covers” and “scans” to the list of additional file matching patterns.

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.

Selected file naming script

As of Picard version 2.7, multiple file naming scripts are supported. This option allows the user to select the file naming script to use from the list of scripts available. Scripts can be either system preset scripts or user-defined scripts. The available scripts are managed in the [File naming script editor](#) screen, which is displayed when the *Edit script...* button is selected.

Files will be named like this

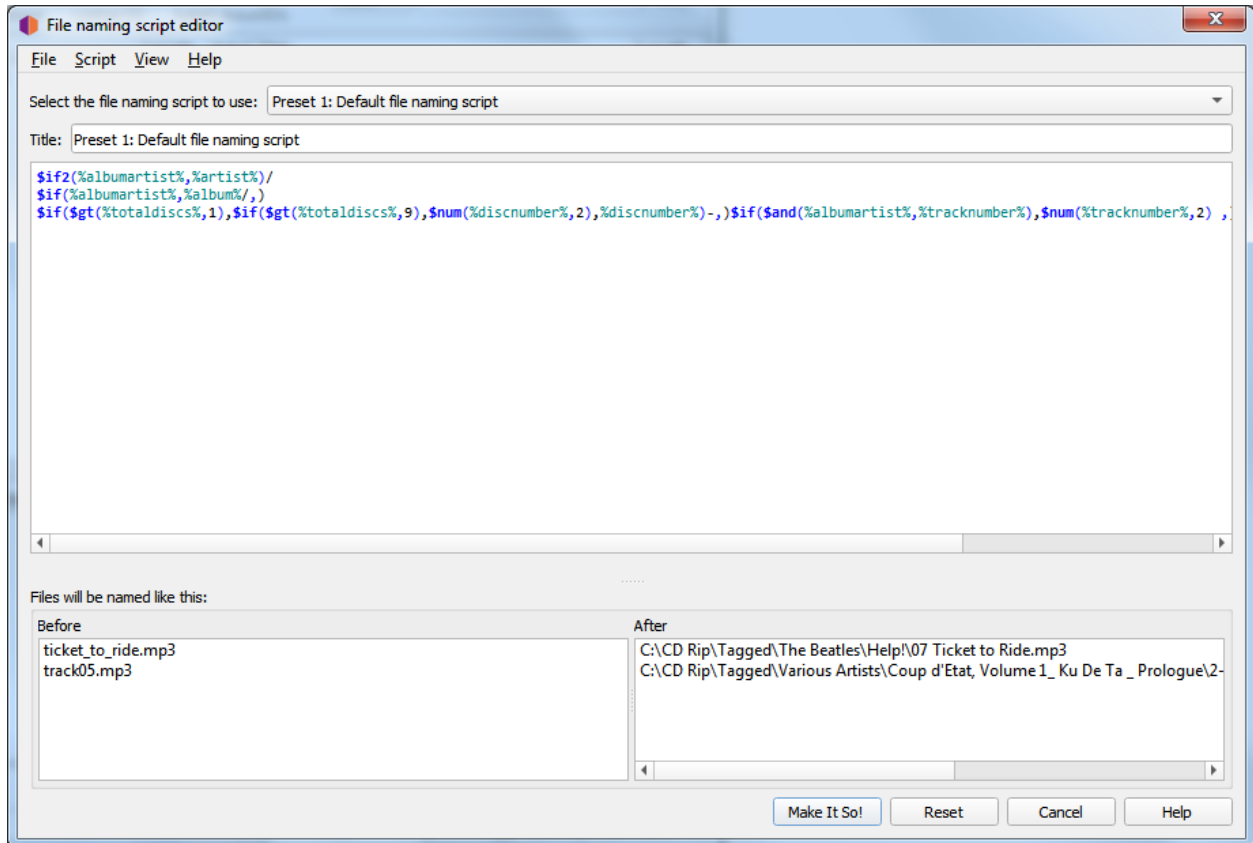
Below the file naming script selector is a section showing examples of the output of the script in two columns: Before and After. If you select files from the Cluster pane or Album pane prior to opening the Options screen, up to 10 files will be randomly chosen from your selection as file naming examples. If you have not selected any files, then some default examples will be provided.

You can change the randomly selected example files from your selected files list by clicking on the *Reload examples* button.

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

File Naming Script Editor

The file naming script editor is used to manage the file naming scripts available for use by Picard. Each script has a title that will show up in the script selection box, and all listed scripts can be edited by the user.



The editor screen has the following sections:

Select the file naming script to use

This option allows the user to select the file naming script to use from the list of scripts available. The selected script will show up in the editing section, where it can be modified if required.

Title

The title assigned to the currently selected script. This can be modified if required.

Script

Below the title is an edit box section containing the formatting string of the selected script. This 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 generally referred to as a “file naming script”, and is in *Picard’s scripting language*.

The script editor automatically highlights the elements of the script, including *function names* and *tag and variable names*. Hovering your mouse pointer over one of the highlighted entries will display help information about the entry if available.

Unicode characters can be entered into the script using the format `\uXXXX` where “XXXX” is the hexadecimal value of the unicode character. It is not recommended to include unicode characters in the directory or filename.

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.

Each file naming script 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.

Scripts are often discussed in the [MetaBrainz Community Forum](#), and there is a thread specific to [file naming and script snippets](#). There is also a tutorial on [Writing a File Naming Script](#) available.

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

Files will be named like this

Below the file naming script is a section showing examples of the output of the script in two columns: Before and After. If you select files from the Cluster pane or Album pane prior to opening the Options screen, up to 10 files will be randomly chosen from your selection as file naming examples. If you have not selected any files, then some default examples will be provided.

Menu bar

At the top of the screen is a menu bar that provides script management functions as well as display settings options. The display settings include:

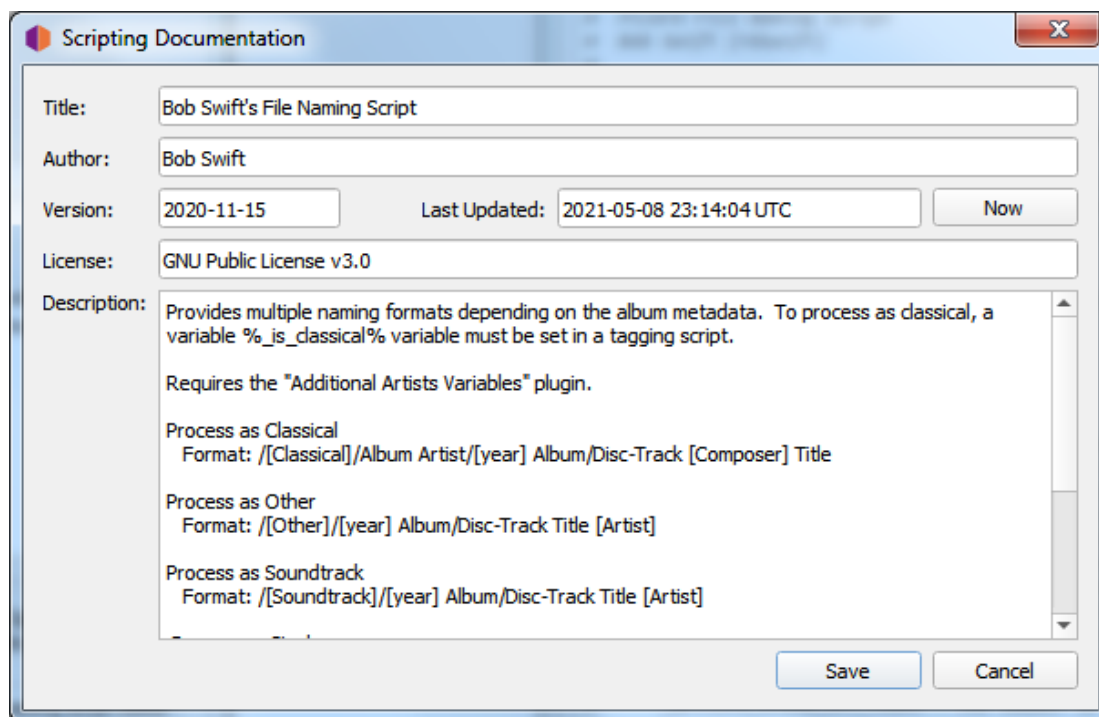
- Word wrap script - This will toggle word wrap on and off in the script edit box.
- Show help tooltips - This will determine whether or not the information is displayed when hovering over a highlighted item.
- Show documentation - This will toggle displaying the scripting documentation in a sidebar on the screen.

There is also an option to update the randomly selected example files from your selected files list.

The script management functionality includes:

- Import a new script from a file, either as a plain-text script or a Picard Naming Script Package.
- Export the current script to a file, either as a plain-text script or a Picard Naming Script Package.
- Add a new (default) script. This can be a blank script or one of the basic system preset scripts provided by Picard.
- Copy the current script as a new script.
- Delete the current script.
- Reset all scripts, also available via the *Reset* button.
- Save all changes, also available via the *Make It So!* button.
- Exit without saving changes, also available via the *Cancel* button.

Script Metadata

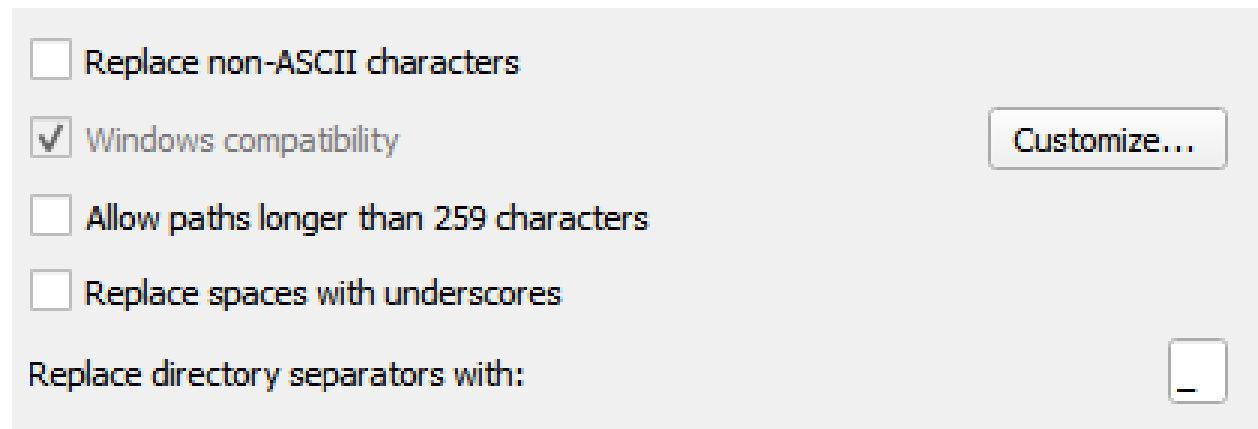


The screenshot shows a window titled "Scripting Documentation" with a close button (X) in the top right corner. The window contains several text input fields and a large text area for a description. The fields are: Title (Bob Swift's File Naming Script), Author (Bob Swift), Version (2020-11-15), Last Updated (2021-05-08 23:14:04 UTC), and License (GNU Public License v3.0). There is a "Now" button next to the Last Updated field. The Description field contains the following text: "Provides multiple naming formats depending on the album metadata. To process as classical, a variable %_is_classical% variable must be set in a tagging script. Requires the 'Additional Artists Variables' plugin. Process as Classical Format: /[Classical]/Album Artist/[year] Album/Disc-Track [Composer] Title Process as Other Format: /[Other]/[year] Album/Disc-Track Title [Artist] Process as Soundtrack Format: /[Soundtrack]/[year] Album/Disc-Track Title [Artist]". At the bottom right of the window are "Save" and "Cancel" buttons.

There is also an option to view/edit the metadata details for the current script. These details include such things as title, author, version, license, description, and date and time of the last update. It is recommended that the description include things such as any required plugins, settings, or tagging scripts. This can also be triggered by double clicking the script title text box.

This information is saved in the Picard Naming Script Package file, and is included when a script package file is imported.

File Naming Compatibility Options



The dialog box contains the following options:

- ☐ Replace non-ASCII characters
- ☒ Windows compatibility
- ☐ Allow paths longer than 259 characters
- ☐ Replace spaces with underscores

Below these options is a label "Replace directory separators with:" followed by a small text input field containing an underscore character. To the right of the first two options is a "Customize..." button.

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

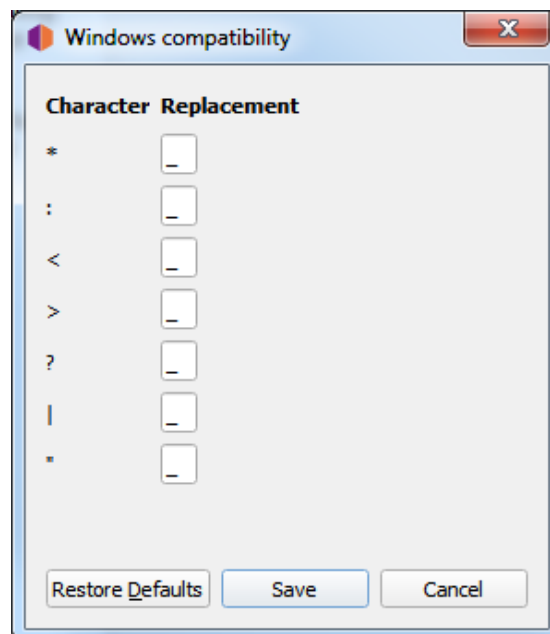
Replace non-ASCII characters

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

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.

As of version 2.9 Picard allows the user to specify what replacement characters to use as replacements for selected characters.



Allow paths longer than 259 characters

This option allows the user to disable the 259 character path limit Picard would usually enforce in Windows compatibility mode when renaming and/or moving files. This is possible both on Windows and on other platforms with Windows compatibility enabled.

Warning: Enabling long paths on Windows might cause files being saved with path names exceeding the 259 character limit traditionally imposed by the Windows API. Some software might not be able to properly access those files. In particular Windows Explorer cannot rename files with long path names or create new files inside folders if the resulting path length would exceed the length limit.

Replace spaces with underscores

When enabled, this option directs Picard to replace all spaces with underscores in the file path and name generated by the selected file naming script.

Replace directory separators with

By default Picard will replace any path separators (slash or backslash) with an underscore when using a tag or variable value as part of the file path and name generated by the selected file naming script. This option allows the user to set a different replacement character other than the underscore.

6.3.7 Fingerprinting Options

Audio Fingerprinting

☐ Do not use audio fingerprinting

☒ Use AcoustID

AcoustID Settings

☒ Ignore existing AcoustID fingerprints

☐ Save AcoustID fingerprints to file tags

Maximum threads to use for calculator: 2

Fingerprint calculator:

C:\Program Files\MusicBrainz Picard\fpalc.exe Browse... Download...

fpalc version 1.5.1 (FFmpeg Lavc58.134.100 Lavf58.76.100 SwR3.9.100)

API key:

Get API key...

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](#) (an Open Source [acoustic fingerprinting](#) system created by [Lukáš Lalinský](#)) but has previously supported TRM and MusicID PUID.

Audio Fingerprinting

This allows you to select whether or not to enable acoustic fingerprinting within Picard. If acoustic fingerprinting is disabled 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.

Save AcoustID fingerprints to file tags

When checked, the AcoustID fingerprint information from scanned files will be saved to the `acoustid_fingerprint` tag. Note that this option is disabled by default because the fingerprint can always be calculated again from the

audio file, and it can add a rather long data tag to the file. The option to save this information has been added as of Picard v2.7 to accommodate use cases such as a workflow where the user adds this tag directly after ripping to avoid having to redo the calculation in the future.

Maximum threads to use for calculator

This allows you to specify the maximum number of fingerprint calculations to be run concurrently. The default value is 2.

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](#) (**fpca1c**) utility which is included with the Picard installation.

Picard will auto-detect the path unless you have specifically overwritten it with something different than the detected path. “*Options → Fingerprinting*” will show the auto detected path as a placeholder hint in the text input, and also uses it for validating the executable. The user only needs to actually select something if auto detection does not work. If the user has selected a specific path this will be used.

API key

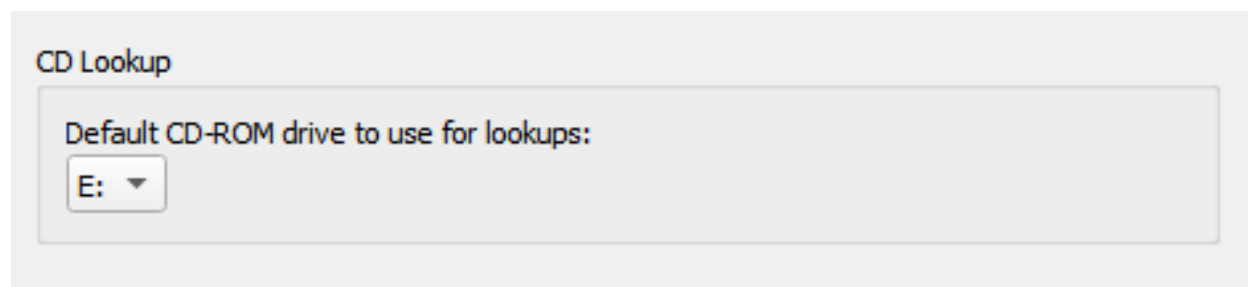
The key used to access the AcoustID API to lookup and submit AcoustID fingerprints. There is no cost to obtain an API key.

6.3.8 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.

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, the CD Lookup 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 for the CD Lookup option. 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.

6.3.9 Plugins Options

Plugins

Name	Version	Actions
Abbreviate artist-sort	0.4.1	
AcousticBrainz Tags	2.2.2	
AcousticBrainz Tonal-Rhythm	1.1.5	
Add Cluster As Release	0.7.3	
Additional Artists Details	0.3	
Additional Artists Variables	0.8.1 → 0.8.2	
Album Artist Website	1.1	
AlbumArtist Extension	0.6.1	
Amazon cover art	1.1	
BPM Analyzer	1.5.1	

Install plugin...Open plugin folderReload List of Plugins

Details

New version available: 0.8.2

This plugin provides specialized album and track variables for use in naming scripts. It is based on the "Album Artist Extension" plugin, but expands the functionality to also include track artists. Note that it cannot be used as a direct drop-in replacement for the "Album Artist Extension" plugin because the variables are provided with different names. This will require changes to existing scripts if switching to this plugin.

Please see the [user guide](#) on GitHub for more information.

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](#) on the Picard website.

There are also a number of plugins available by third-party developers. Often these are discussed on the [Community Discussion Forum](#) 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 Ray](#) maintains an unofficial list of available plugins on a [wiki page](#). 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.

Plugins

Name	Version	Actions
Add Cluster As Release	0.7.3	
Additional Artists Details	0.3	
Additional Artists Variables	0.8.1 → 0.8.2	
Album Artist Website	1.1	

Install plugin...Open plugin folderReload List of Plugins

Details

New version available: 0.8.2

This plugin provides specialized album and track variables for use in naming scripts. It is based on the "Album Artist Extension" plugin, but expands the functionality to also include track artists. Note that it cannot be used as a direct drop-in replacement for the "Album Artist Extension" plugin because the variables are provided with different names. This will require changes to existing scripts if switching to this plugin.

Please see the [user guide](#) on GitHub for more information.

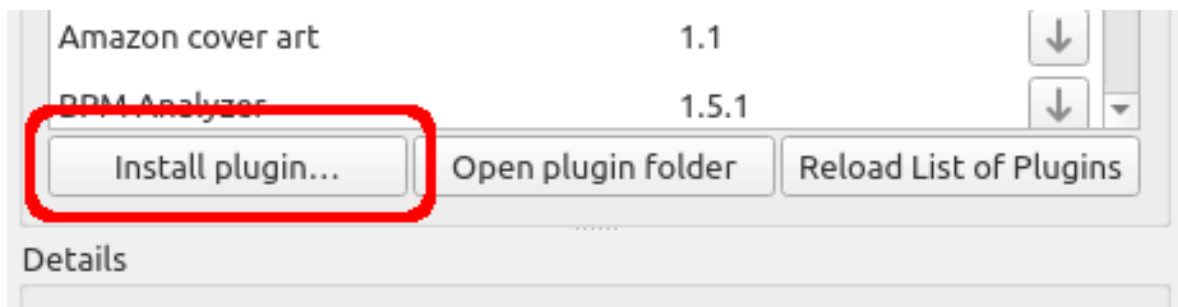
Name: Additional Artists Variables
Authors: Bob Swift (rdswift)
License: GPL-2.0-or-later
Files: additional_artists_variables.zip/additional_artists_variables.py
User Guide: https://github.com/rdswift/picard-plugins/blob/2.0_RDS_Plugins/plugins/additional_artists_variables/docs/README.md

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.

6.3.10 User Interface Options

Appearance

- ☒ Show text labels under icons
- ☒ Show icons in menus
- User interface language:
- User interface color theme:

Miscellaneous

- ☐ Allow selection of multiple directories
- ☒ Use builtin search rather than looking in browser
- ☐ Use advanced query syntax
- ☐ Show the new user dialog when starting Picard
- ☒ Show a quit confirmation dialog for unsaved changes
- ☒ Show a confirmation dialog when saving files
- ☒ Adjust horizontal position in file browser automatically
- ☒ Begin browsing in the following directory:

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.

Show icons in menus

Some users prefer to disable menu icons, which is the default behavior for macOS systems. This option allows the user to select whether the icons are displayed in the menus.

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.

User interface color theme

This option allows the user to select the color theme used by Picard. On macOS and Windows systems, the available choices are:

- Default - The default color scheme based on the operating system display settings.
- Light - A light display theme.
- Dark - A dark display theme.

On Linux and similar operating systems, the available choices are:

- Default - The default color scheme based on the operating system display settings.
- System - The Qt5 theme configured in the desktop environment.

Note: The colors for the light and dark themes can be customized in the [Colors](#) section. Separate sets of color selections are maintained for the light and dark themes. The colors for the currently displayed theme are the ones displayed for editing.

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.

Warning: When enabling the multiple directories option setting Picard will no longer use the system file picker for selecting directories. This may result in reduced functionality.

Use built-in search rather than looking in browser

When this option is enabled the search for albums, artists or tracks will show the results in a dialog. By default this option is enabled. If this option is disabled Picard will open a search on MusicBrainz.org in your default web browser.

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 the new user dialog when starting Picard

When this is enabled, Picard will show a dialog intended for new users when you start the program. This displays a warning about the consequences of saving files, along with a suggestion for minimizing the impact until you have confirmed that your configuration produces the expected results. It also provides a link to the on-line documentation.

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.

Show a confirmation dialog when saving files

When this is enabled, Picard will show a dialog when you save files, indicating what actions will be performed on the files and the number of files to be saved. This may help prevent accidentally making changes that you are not expecting.

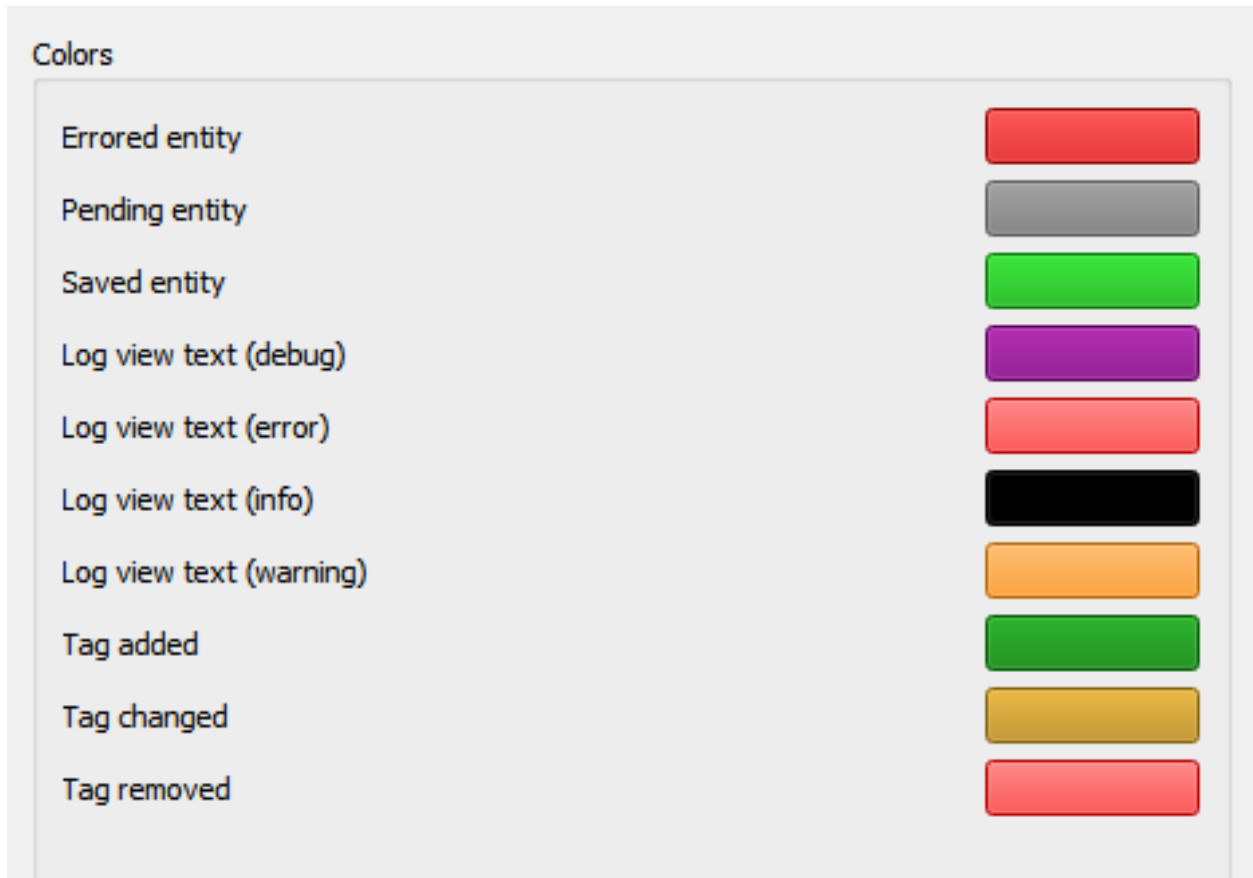
Adjust horizontal position in file browser automatically

When this is enabled, Picard will automatically scroll the file browser display to keep relevant content visible.

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.

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

Note: Separate sets of color selections are maintained for the light and dark themes. The colors for the currently displayed theme are the ones displayed for editing.

Top Tags

Show the below tags above all other tags in the metadata view

title

artist

album

tracknumber

~length

date

Add new tag

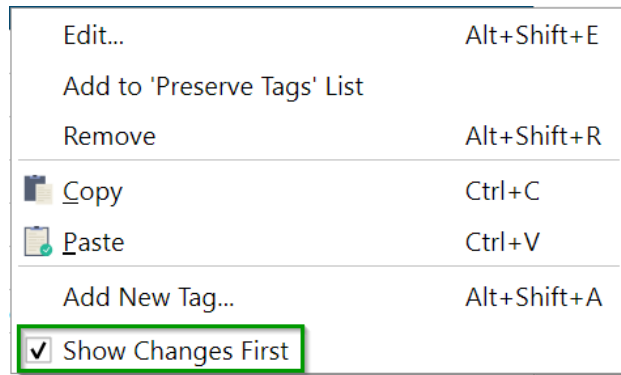
^

▼

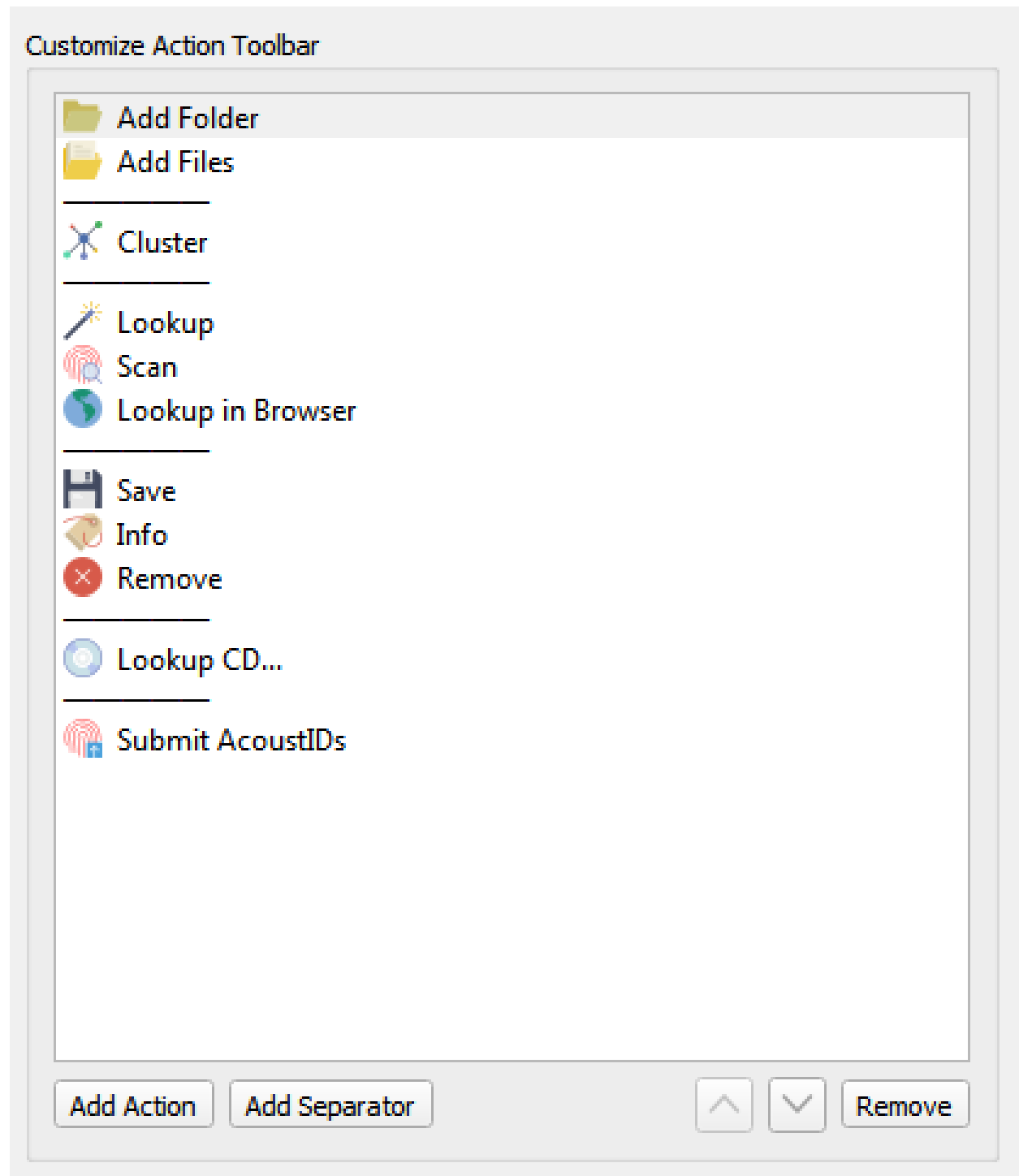
Remove tags

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.

Note: By default, Picard will display the top tags configured here first in the list. If you right click on one of the tags in the metadata pane, and enable *“Show Changes First”* in the context menu, the tags with changes will always be displayed first in the list, followed by the remaining top tags and other tags.



Toolbar



Customize Action Toolbar

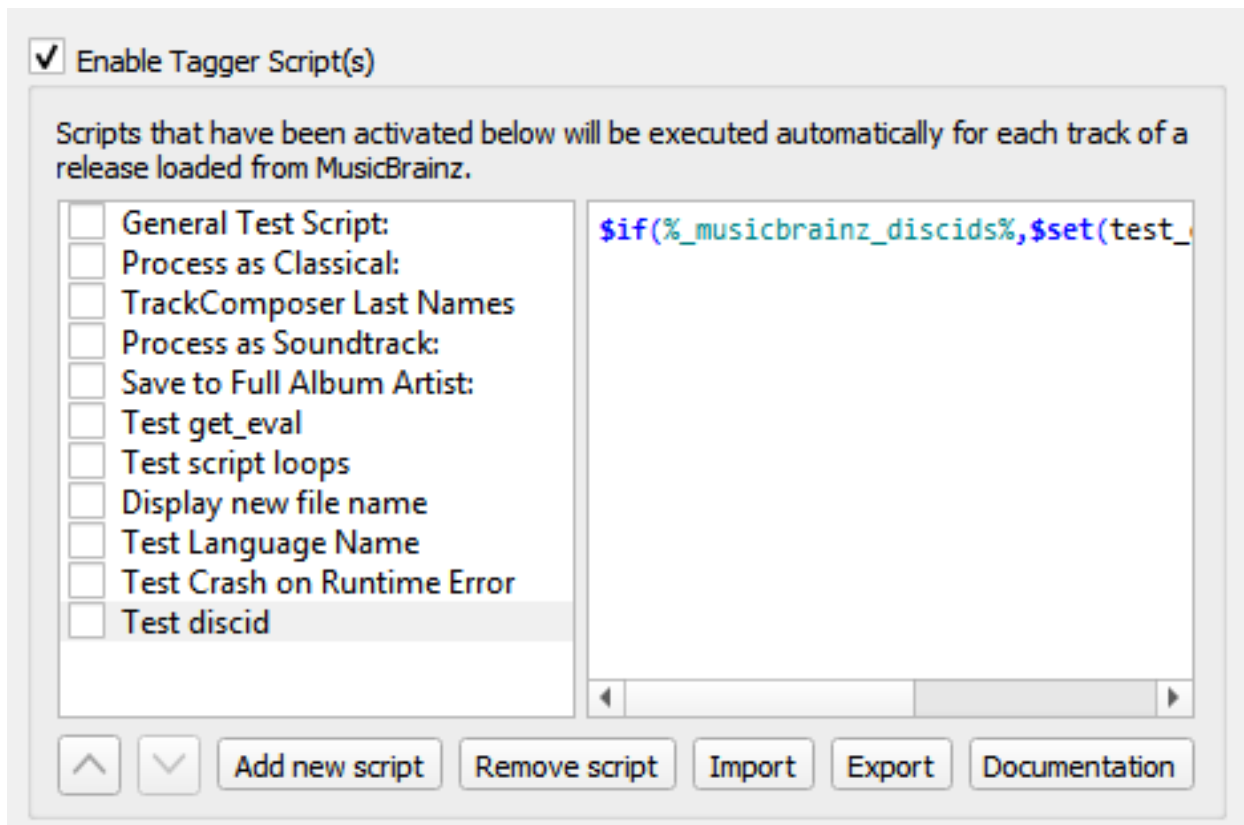
This allows you to to add, remove or rearrange the items displayed in the Action Toolbar.

Items that can be included in the toolbar include:

- Add Files
- Add Folder
- Cluster
- Generate Fingerprints
- Info
- Lookup
- Lookup CD...
- Lookup in Browser
- Open in Player
- Parse File Names...
- Remove
- Save
- Scan
- Submit AcoustIDs

In addition, separators can be added to allow grouping of items.

6.3.11 Scripting Options



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

The “Enable 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.

As of Picard v2.7 you can also import a new script from a file, or export an existing script to a file. Files can be stored as either a plain-text script or a Picard Tagging Script Package stored in YAML format.

The script editor automatically highlights the elements of the script, including *function names* and *tag and variable names*. Hovering your mouse pointer over one of the highlighted entries will display help information about the entry if available.

Unicode characters can be entered into the script using the format `\uXXXX` where “XXXX” is the hexadecimal value of the unicode character.

When the checkbox beside the script is checked, that script will be executed automatically, once for each track in the release, when Picard retrieves information for a release

from the MusicBrainz website. If the checkbox is left unchecked, then the script will not be executed automatically.

Regardless of whether or not the script is executed automatically, it can also be executed manually by right-clicking on an item in the clustering pane (middle pane) or the tagging pane (right-hand pane) and selecting it from the list displayed when “*Run Scripts*” is selected. If a cluster is selected in the middle pane or a release is selected in the right-hand pane, the script will be executed for each track in the selected cluster or release. If only a single track or file is selected, then the script will only be executed for that track or file.

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

6.3.12 Advanced Options

Advanced options

Ignore file paths matching the following regular expression:

☐ Ignore hidden files
☒ Include sub-folders when adding files from folder
Ignore track duration difference under this number of seconds
Maximum number of entities to return per MusicBrainz query

Ignore the following tracks when determining whether a release is complete

☐ Video tracks
☐ Pregap tracks
☐ Data tracks
☐ Silent tracks

Tags to ignore for comparison:

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.

Maximum number of entities to return per MusicBrainz query

This sets the maximum number of results returned for queries made to the MusicBrainz website. The default value is 50 results. On Picard v2.8.1 and earlier, this value was fixed at a maximum of 25 responses.

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

The screenshot shows the 'Network' settings window in MusicBrainz Picard. At the top, there is a checkbox for 'Web Proxy' which is currently unchecked. Below it, there are two radio buttons for 'HTTP' (selected) and 'SOCKS'. Under the 'HTTP' section, there are three input fields: 'Server address:', 'Port:' (with a spinner box set to 80), 'Username:', and 'Password:'. Below these fields is a 'Request timeout in seconds:' label with a spinner box set to 30. Further down, there is a checked checkbox for 'Browser Integration'. Under this, there is a 'Default listening port:' label with a spinner box set to 8000, and a checked checkbox for 'Listen only on localhost'.


Web Proxy

If you need a proxy to make an outside network connection you may specify one here. You can choose between HTTP and SOCKS proxy. The required settings are **Server Address** and **Port**. If the proxy requires authentication also enter **Username** and **Password**.

Request timeout in seconds

By default Picard will abort running network requests after 30 seconds of inactivity. If needed you can change the timeout period here.

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  next to releases and recordings. Clicking on this button will load the corresponding release or recording into Picard.

Default listening port

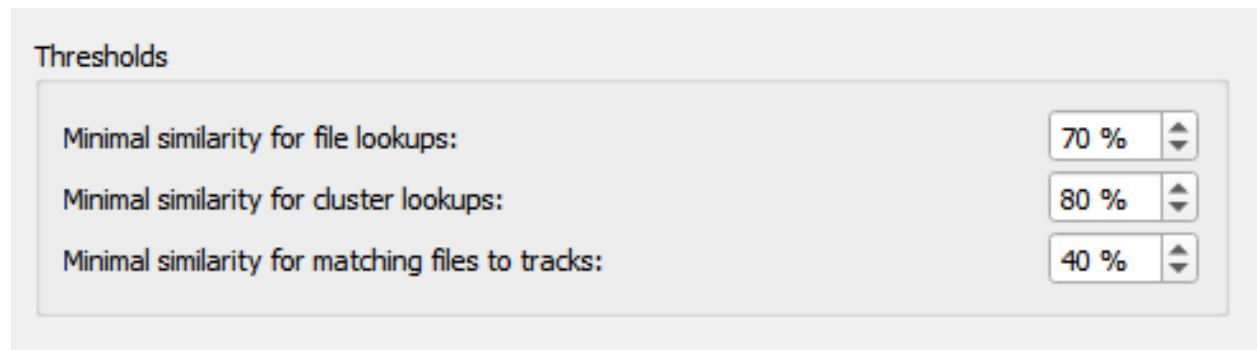
This identifies the default port Picard will listen on for the browser integration. If the port is not available Picard will try to increase the port number by one until it finds a free port.

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](#) 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.

Matching



The screenshot shows a window titled "Thresholds" with three settings, each with a text label and a percentage spinner control:

Setting	Value
Minimal similarity for file lookups:	70 %
Minimal similarity for cluster lookups:	80 %
Minimal similarity for matching files to tracks:	40 %

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.

This setting is used when you do a lookup on individual files. The files' metadata is compared to the recordings found on MusicBrainz. If the similarity is below the threshold the recording is ignored completely.

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.

This setting is used when a file is assigned to a release and Picard needs to decide which track to assign it to. This happens on cluster lookup after the release has been chosen, or if you manually drag files onto a loaded release (as opposed to dragging it onto a track directly). If none of the tracks gives a match above the threshold the file is moved into an “unmatched files” section on that release.

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.

Maintenance

Configuration File:

C:/Development/MusicBrainz/Picard/PicardTesting.ini

Open folder

Load Backup

Save Backup

The configuration file currently contains 171 option settings, 17 which are unused.

☒ Remove selected options

This allows you to remove unused option settings from the configuration INI file.

Settings that are found in the configuration file that do not appear on any option settings page will be listed below. If your configuration file does not contain any unused option settings, then the list will be empty and the removal checkbox will be disabled.

Note that unused option settings could come from plugins that have been uninstalled, so please be careful to not remove settings that you may want to use later when the plugin is reinstalled. Options belonging to plugins that are installed but currently disabled will not be listed for possible removal.

To remove one or more settings, first enable the removal by checking the "Remove selected options" box. You can then select the settings to remove by checking the box next to the setting. When you choose "Make It So!" to save your option settings, the selected items will be removed.

☐ Select all

Option	Value
<input type="checkbox"/> artist_locale	'en'
<input type="checkbox"/> artist_script_exceptions	[('LATIN', 0)]
<input type="checkbox"/> ca_provider_use_amazon	'true'
<input type="checkbox"/> ca_provider_use_caa	'true'
<input type="checkbox"/> ca_provider_use_caa_release_group_fallback	'true'
<input type="checkbox"/> ca_provider_use_whitelist	'true'
<input type="checkbox"/> file_naming_scripts	{'7c3bba43-3ae8-4ff5-97a9-80b534380ce0': {'author': 'Bob Swift', 'deletable': True, 'description': 'Provides multiple naming formats depending on

Configuration File

This displays the path and file name of the configuration file currently used by Picard. This file contains all of your settings and preferences, and the format of the file is consistent across operating systems.

The *Save Backup* button allows you to create a backup copy of the current configuration file. This can be used to easily copy the settings for use on a different computer, or to provide a snapshot for easy recovery in the event that the configuration becomes corrupted or you want to undo configuration changes. It also allows you to have different configurations available without using profiles.

The *Load Backup* button allows you to replace the current configuration file with a backup created earlier. Loading a backup configuration file will replace all of the current configuration settings. Prior to loading the file, Picard will automatically save a backup copy of the current file.

Configuration File Cleanup

Over the course of trying out plugins and such, the configuration file can become bloated with orphaned settings. This section allows you to remove unused option settings from the configuration file.

Settings that are found in the configuration file that do not appear on any option settings page will be listed. If your configuration file does not contain any unused option settings, then the list will be empty and the removal checkbox will be disabled.

To remove one or more settings, first enable the removal by checking the “Remove selected options” box. You can then select the settings to remove by checking the box next to the setting. When you choose *Make It So!* to save your option settings, the selected items will be removed.

Note: Unused option settings could come from plugins that have been uninstalled, so please be careful to not remove settings that you may want to use later when the plugin is reinstalled. Options belonging to plugins that are installed but currently disabled will not be listed for possible removal.

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](#) on the website.

7.1 Basic Tags

The following tags are supported and are written in the standard format that can be read by other software. The information is populated from MusicBrainz data for most releases, without any special Picard settings. Note that some of the information such as 'lyrics' is not contained within the MusicBrainz database, and will not be provided automatically.

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%`).

Some tags provide the MusicBrainz Identifier (MBID) of the entity. The MBID is a 32-character identifier assigned to an entity (e.g.: artist, album, track or work) to uniquely identify the entity. For more information about MBIDs, please see the [MusicBrainz Identifier](#) page in the MusicBrainz documentation.

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](#) section for details.

7.1.1 Tags Provided from MusicBrainz Data

These tags will be provided based on the information from the MusicBrainz database and will be populated automatically by Picard if the information is available.

album

The title of the release.

albumartist

The artists primarily credited on the release, separated by the specified join phrases. These could be either “standardized” or “as credited” depending on whether the “Use standardized artist names” metadata option is enabled.

albumartistsort

The release artists sort names, separated by the specified join phrases. (e.g.: “Beatles, The”)

artist

The track artist names, separated by the specified join phrases. These could be either “standardized” or “as credited” depending on whether the “Use standardized artist names” metadata option is enabled.

artists

A multi-value tag containing the track artist names. These could be either “standardized” or “as credited” depending on whether the “Use standardized artist names” metadata option is enabled. (*since Picard 1.3*)

artistsort

The track artists sort names, separated by the specified join phrases.

asin

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

barcode

The barcode assigned to the release.

catalognumber

A multi-value tag containing the numbers assigned to the release by the labels, 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

The disambiguation comment entered to help distinguish one release from another (e.g.: Deluxe version with 2 bonus tracks). This is not populated by stock Picard. It may be used and populated by certain plugins. Picard stores this information in the **_releasecomment** variable.

compilation

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

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

date

The date that the release (album) was issued, in the format YYYY-MM-DD.

discnumber

The number of the disc in the release that contains this track.

discsubtitle

The media title given to a specific disc in the release.

isrc

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

label

A multi-value tag containing the names of the labels associated with the release.

media

The media on which the release was distributed (e.g.: CD). See the [Release Format](#) page on the MusicBrainz website for more information.

musicbrainz_albumartistid

A multi-value tag containing the MBIDs for the release artists.

musicbrainz_albumid

The MBID for the release.

musicbrainz_artistid

A multi-value tag containing the MBIDs for the track artists.

musicbrainz_discid

The 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. This tag contains the Disc ID if the album information was retrieved using “Tools → Lookup CD”. *(since Picard 0.12)*

musicbrainz_originalalbumid

The MBID for the original release. This is only available if the release has been merged with another release.

musicbrainz_originalartistid

A multi-value tag containing the MBIDs for the track artists of the original recording. This is only available if the recording has been merged with another recording.

musicbrainz_recordingid

The MBID for the recording.

musicbrainz_releasegroupid

The MBID for the release group.

musicbrainz_trackid

The MBID for the track.

originaldate

The original release date in the format YYYY-MM-DD. By default this is set to the earliest release in the release group. This can 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: This is the same information provided in the `_releasegroup_firstreleasedate` variable, and is consistent across all tracks in the release. If you prefer to have this tag populated with the date of the earliest recording of the track in the database, which will likely be different for each track in the release, this can be achieved by enabling a one-line tagging script as `$set(originaldate,%_recording_firstreleasedate%)`. Be aware that setting this can cause a release to be scattered across multiple directories if you use `%originaldate%` as part of the path portion of your file naming script.

Note: If you are storing tags in MP3 files as ID3v2.3 then the original date can only be stored as a year.

originalyear

The year of the original release date in the format YYYY. By default this is set to the earliest release in the release group. This can provide, for example, the release year of the vinyl version of what you have on CD.

releasecountry

The two-character code for the country in which the release was issued. If more than one release country was specified, this tag will contain the first one in the list.

releasestatus

An indicator of the “official” status of the release. Possible values include *official*, *promotional*, *bootleg*, and *pseudo-release*.

releasetype

A multi-value tag containing the types of release assigned to the release group. 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](#) standard. (*since Picard 0.10*)

title

The title of the track.

totaldiscs

The total number of discs in this release.

totaltracks

The total number of tracks on this disc.

tracknumber

The number of the track on the disc.

website

The official website for the artist.

7.1.2 Tags Not Provided from MusicBrainz Data

These tags are not able to be populated by stock Picard, however they may be used and populated by certain plugins or scripts.

acoustid_fingerprint

The Acoustic Fingerprint for the track. The fingerprint is based on the audio information found in a file, and is calculated using the [Chromaprint](#) software.

acoustid_id

The AcoustID associated with the track. The AcoustID is the identifier assigned to an audio file based on its acoustic fingerprint. Multiple fingerprints may be assigned the same AcoustID if the fingerprints are similar enough. See the section on [Understanding Acoustic Fingerprinting and AcoustIDs](#) for more information.

albumsort

The sort name of the title of the release.

bpm

The number of beats per minute of the track.

copyright

The copyright message for the copyright holder of the original sound, beginning with a year and a space character.

encodedby

The person or organization that encoded the track.

encodersettings

The settings used when encoding the track.

key

The key of the music.

lyrics

The lyrics for the track.

musicip_fingerprint

The MusicIP Fingerprint for the track.

musicip_puid

The MusicIP PUIDs associated with the track.

originalalbum

The release title of the earliest release in the release group intended for the title of the original recording.

originalartist

The track artist of the earliest release in the release group intended for the performers of the original recording.

originalfilename

The original name of the audio file.

releasedate

Explicit tag for the release date (*since Picard 2.9*). This tag exists for specific use in scripts and plugins, but is not filled by default. In most cases it is recommended to use the date tag instead for compatibility with existing software.

showmovement

The work and movement of the track.

subtitle

This is used for information directly related to the contents title.

titlesort

The sort name of the track title.

7.1.3 iTunes-Specific Tags

These tags are only available in iTunes files and are not able to be populated by stock Picard, however they may be used and populated by certain plugins or scripts.

gapless

Indicates whether or not there are gaps between the recordings on the release.

podcast

Indicates whether or not the recording is a podcast.

podcasturl

The associated url if the recording is a podcast.

show

The name of the show if the recording is associated with a television program.

showsort

The sort name of the show if the recording is associated with a television program.

7.2 Advanced Tags

You can make additional tags available by enabling the [Use track relationships](#) and/or the [Use genres from MusicBrainz](#) settings in Picard.

Some tags provide the MusicBrainz Identifier (MBID) of the entity. The MBID is a 32-character identifier assigned to an entity (e.g.: artist, album, track or work) to uniquely identify the entity. For more information about MBIDs, please see the [MusicBrainz Identifier](#) page in the MusicBrainz documentation.

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](#) section for details.

7.2.1 Track Relationship Tags

If you enable “Use track relationships” in the Option settings, you get these extra tags:

arranger

The names of the arrangers associated with the track. These can include the instrument and orchestra arrangers, and could be associated with the release, recording or work. (*since Picard 0.10*)

composer

The names of the composers for the associated work.

composersort

The sort names of the composers for the associated work.

conductor

The names of the conductors associated with the track. These can include the conductor and chorus master, and could be associated with the release or recording.

director

The director of a track as provided by the *Video Director* or *Audio Director* relationship in MusicBrainz. (*Since Picard 2.6, updated in Picard 2.9*)

djmixer

The names of the DJ mixers for the track. (*since Picard 0.9*)

engineer

The names of the engineers associated with the track.

language

Work lyric language as per [ISO 639-3](#) if a related work exists. (*since Picard 0.10*)

license

The licenses associated with the track, either through the release or recording relationships. (*since Picard 1.0*)

lyricist

The names of the lyricists for the associated work.

mixer

The names of the “*Mixed By*” engineers associated with the track. (*since Picard 0.9*)

musicbrainz_workid

The MBID for the Work if a related work exists.

performer:<type>

The names of the performers for the specified type. These types include:

- vocals or instruments for the associated release or recording, where <type> can be “*vocal*”, “*guest guitar*”, “*solo violin*”, etc.
- the orchestra for the associated release or recording, where <type> is “*orchestra*”
- the concert master for the associated release or recording, where <type> is “*concertmaster*”

producer

The names of the producers for the associated release or recording.

remixer

The names of the remixer engineers associated with the track.

work

The name of the work associated with the track. (*since Picard 1.3*)

writer

A multi-value tag containing the names of the writers associated with the related work. (*since Picard 1.0*). This is not written to most file formats automatically. You can merge this with composers with a script like:

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

Note: Some tags such as **performer** are available as both track and release level relationships, and the values may be different depending on which relationship options are enabled.

7.2.2 Genre Tags

If you enable “Use genres from MusicBrainz”, you get:

genre

A multi-value tag containing the specified genre information from MusicBrainz (*since Picard 2.1, earlier versions used folksonomy tags*)

7.3 Basic Variables

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

Some variables provide the MusicBrainz Identifier (MBID) of the entity. The MBID is a 32-character identifier assigned to an entity (e.g.: artist, album, track or work) to uniquely identify the entity. For more information about MBIDs, please see the [MusicBrainz Identifier](#) page in the MusicBrainz documentation.

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

A multi-value variable containing the names of the album’s artists. These could be either “standardized” or “as credited” depending on whether the “Use standardized artist names” metadata option is enabled. (*since Picard 1.3*)

_albumartists_sort

A multi-value variable containing the sort names of the album’s artists. (*since Picard 1.3*)

_artists_sort

A multi-value variable containing the sort names of the track’s artists. (*since Picard 1.3*)

_datatrack

Set to 1 if the track is a “[data track](#)”, otherwise empty. (*since Picard 1.3.1*)

_discregap

Set to 1 if the disc the track is on has a “pregap track”, otherwise empty. (since Picard 1.4)

_multiartist

Set to 1 if not all of the tracks on the album have the same primary artist, otherwise empty. (since Picard 1.3)

_musicbrainz_discids

A multi-value variable containing a list of all of the disc ids attached to the selected release. The list provided for each medium only includes the disc ids attached to that medium. For example, the list provided for Disc 1 of a three CD set will not include the disc ids attached to discs 2 and 3 of the set.

_musicbrainz_tracknumber

The track number written as on the MusicBrainz release, such as vinyl numbering (A1, A2...).

_pregap

Set to 1 if the track is a “pregap track”, otherwise empty. (since Picard 1.3.1)

_primaryreleasetype

The primary type of the release group (i.e.: *album*, *single*, *ep*, *broadcast*, or *other*).

_rating

The rating of the track from 0-5 by MusicBrainz users.

_recordingcomment

The disambiguation comment for the recording associated with a track. (since Picard 0.15)

_recording_firstreleasedate

The date of the earliest recording for a track in the format YYYY-MM-DD. (Since Picard 2.6)

_releaseannotation

The annotation comment for the release. (since Picard 2.6)

_releasecomment

The disambiguation comment for the release. (since Picard 0.15)

_releasecountries

A multi-value variable containing the complete list of release countries for the release. (since Picard 2.3.1)

_releasegroup

The title of the release group. This is typically the same as the album title, but can be different.

_releasegroup_firstreleasedate

The date of the earliest release in the release group in the format YYYY-MM-DD. This is intended to provide, for example, the release date of the vinyl version of what you have on CD. (*Since Picard 2.6*)

Note: This is the same information provided by default in the `originaldate` tag.

_releasegroupcomment

The disambiguation comment for the release group.

_releaselanguage

The language of the release as per [ISO 639-3](#). (*since Picard 0.10*)

_secondaryreleasetype

Zero or more secondary types (i.e.: *audiobook*, *compilation*, *dj-mix*, *interview*, *live*, *mixtape/street*, *remix*, *soundtrack*, or *spokenword*) for the release group.

_totalalbumtracks

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

7.4 File Variables

These variables are populated from information found in the audio files themselves, without any special Picard settings.

Note: Variables that rely on information from the files (e.g.: `_bitrate`) are only available for use on tracks with attached files, when running scripts manually on files or in the file naming script.

Warning: Prior to version 2.5 Picard did not support using file variables in tagging scripts.

_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*)

_file_created_timestamp

The file creation timestamp in the form 'YYYY-MM-DD HH:MM:SS' as reported by the file system. (*since Picard 2.9*)

_file_modified_timestamp

The file modification timestamp in the form 'YYYY-MM-DD HH:MM:SS' as reported by the file system. (*since Picard 2.9*)

_format

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

_length

The length of the track in format mins:secs.

_sample_rate

Number of digitizing samples per second (Hz).

_filesize

Size of the file in bytes.

7.5 Advanced Variables

You can make additional tags available by enabling the [Use track relationships](#) and/or the [Use release relationships](#) settings in Picard.

Some variables provide the MusicBrainz Identifier (MBID) of the entity. The MBID is a 32-character identifier assigned to an entity (e.g.: artist, album, track or work) to uniquely identify the entity. For more information about MBIDs, please see the [MusicBrainz Identifier](#) page in the MusicBrainz documentation.

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

7.5.1 Release Relationship Variables

If you enable tagging with *Use release relationships*, you get these extra variables:

`_release_series`

A multi-value variable containing the series titles associated with the release. *(since Picard 2.9)*

`_release_seriesid`

A multi-value variable containing the series MBIDs associated with the release. *(since Picard 2.9)*

`_release_seriescomment`

A multi-value variable containing the series disambiguation comments associated with the release. *(since Picard 2.9)*

`_release_seriesnumber`

A multi-value variable containing the series numbers associated with the release. *(since Picard 2.9)*

`_releasegroup_series`

A multi-value variable containing the series titles associated with the release group. *(since Picard 2.9)*

`_releasegroup_seriesid`

A multi-value variable containing the series MBIDs associated with the release group. *(since Picard 2.9)*

`_releasegroup_seriescomment`

A multi-value variable containing the series disambiguation comments associated with the release group. *(since Picard 2.9)*

`_releasegroup_seriesnumber`

A multi-value variable containing the series numbers associated with the release group. *(since Picard 2.9)*

7.5.2 Track Relationship Variables

If you enable tagging with *Use track relationships*, you get these extra variables:

`_lyricistsort`

The sort names of the lyricists for the work. *(since Picard 2.9)*

`_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)
```

_recordingtitle

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

_recording_series

A multi-value variable containing the series titles associated with the recording. (since Picard 2.9)

_recording_seriesid

A multi-value variable containing the series MBIDs associated with the recording. (since Picard 2.9)

_recording_seriescomment

A multi-value variable containing the series disambiguation comments associated with the recording. (since Picard 2.9)

_recording_seriesnumber

A multi-value variable containing the series numbers associated with the recording. (since Picard 2.9)

_workcomment

The disambiguation comment associated with the work. (since Picard 2.7)

_work_series

A multi-value variable containing the series titles associated with the work. (since Picard 2.9)

_work_seriesid

A multi-value variable containing the series MBIDs associated with the work. (since Picard 2.9)

_work_seriescomment

A multi-value variable containing the series disambiguation comments associated with the work. (since Picard 2.9)

_work_seriesnumber

A multi-value variable containing the series numbers associated with the work. (since Picard 2.9)

_writersort

The sort names of the writers for the work. (since Picard 2.9)

7.6 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.

7.7 Tags from Plugins

Plugins from Picard [Plugins](#) can add more tags. Following are some examples.

7.7.1 Last.fm Plugin

genre

Pseudo-genre based on folksonomy tags.

7.7.2 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](#) project on GitHub for additional information.

7.8 Other Information

For technical details on how tags are written into files, see the *Picard Tag Mapping* section.

If you set variables that are not known to Picard, these will be saved as new tags in ID3, MP4, APEv2 and Vorbis based files. They will not be saved in ASF based files.

- For ID3 based files these tags will be saved to, and reloaded from, ID3 user defined text information (TXXX) frames.
- For MP4 files these tags will be saved with a prefix of `----:com.apple.iTunes:.` This is widely understood by other tools to be used for custom tags.
- For Vorbis and APEv2 files these tags will be saved as given.

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:`, e.g. `%_id3:TXXX:mytag%`. Currently these tags are not loaded into variables when you reload the file into Picard (*since Picard 0.9*).

Note: Saving custom tags to MP4 files is supported since Picard 2.3. Earlier versions will neither save nor load custom tags in MP4 files.

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](#), and there is a thread specific to [file naming and script snippets](#).

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.

8.1 Syntax

The syntax is derived from [Foobar2000’s titleformat](#). 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(...)).

Note: When entering input strings into Picard scripts you have to escape a backslash “\”, dollar sign “\$”, comma “,” and the left and right parentheses “(” and “)” in order to force Picard to not interpret them as part of the script command. This is done by inserting a backslash before the character to be escaped. For example, to set a tag value to (\$1,000,000) it would have to be entered as `$set(test_tag,\(\ $1\,000\,000\))`.

Note: Usually you can access the values of a tag by the proper variable name. For example, if your tag is called “rerecorded” you can use %rerecorded%. But the hyphen is not a valid character for a script variable, so %re-recorded% gives a syntax error. In cases like this you need to use `$get(re-recorded)`.

8.2 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 semi-colon ";", except when used with special multi-value functions.

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

SCRIPTING FUNCTIONS

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

9.1 Assignment Functions

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

9.1.1 `$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: Unlike most functions, in this case the source is specified **without** enclosing it with percent signs (%).

Warning: 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"
```

9.1.2 \$copymerge

Usage: **\$copymerge(target,source[,keep_duplicates])**

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.

If keep_duplicates is set, then the duplicates will not be removed from the result.

Note: Unlike most functions, in this case the source is specified **without** enclosing it with 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)
```

(continues on next page)

(continued from previous page)

```

$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"

$setmulti(target,zero; two)
$setmulti(source,one; two)
$copymerge(target,source,1)  ==>  "zero; two; one; two"

```

9.1.3 \$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
```

9.1.4 \$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
```

9.1.5 \$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")
```

9.1.6 \$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:*)
```

9.2 Text Functions

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

9.2.1 \$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"
```


9.2.2 \$find

Usage: **\$find(haystack,needle)**

Category: text

Implemented: Picard 2.3

Description:

Returns the zero-based index of the first occurrence 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 beginning 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:

<code>\$find(abcdef,a)</code>	<code>==></code>	<code>"0"</code>
<code>\$find(abcdef,c)</code>	<code>==></code>	<code>"2"</code>
<code>\$find(abcdef,cd)</code>	<code>==></code>	<code>"2"</code>
<code>\$find(abcdef,g)</code>	<code>==></code>	<code>" "</code>
<code>\$find(abcdef,B)</code>	<code>==></code>	<code>" "</code>
<code>\$find(,a)</code>	<code>==></code>	<code>" "</code>
<code>\$find(abcdef,)</code>	<code>==></code>	<code>"1"</code>

9.2.3 \$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:

<code>\$firstalphachar(abc)</code>	<code>==></code>	<code>"A"</code>
<code>\$firstalphachar(123)</code>	<code>==></code>	<code>"#"</code>

(continues on next page)

(continued from previous page)

<code>\$firstalphachar(***)</code>	<code>==></code>	<code>"#"</code>
<code>\$firstalphachar(***,)</code>	<code>==></code>	<code>" "</code>
<code>\$firstalphachar(***,^)</code>	<code>==></code>	<code>"^"</code>
<code>\$firstalphachar(***,non-alpha)</code>	<code>==></code>	<code>"non-alpha"</code>

9.2.4 \$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:

<code>\$firstwords(Once upon a time,)</code>	<code>==></code>	<code>" "</code>
<code>\$firstwords(Once upon a time,0)</code>	<code>==></code>	<code>" "</code>
<code>\$firstwords(Once upon a time,3)</code>	<code>==></code>	<code>" "</code>
<code>\$firstwords(Once upon a time,7)</code>	<code>==></code>	<code>"Once"</code>
<code>\$firstwords(Once upon a time,-3)</code>	<code>==></code>	<code>"Once upon a"</code>
<code>\$firstwords(Once upon a time,-30)</code>	<code>==></code>	<code>" "</code>

9.2.5 \$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.

Note: Usually you can access the values of a tag by the proper variable name. For example, if your tag is called "rerecorded" you can use `%rerecorded%`. But the hyphen

is not a valid character for a script variable, so %re-recorded% gives a syntax error. In cases like this you need to use `$get(re-recorded)`.

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)
```

9.2.6 \$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"
```

9.2.7 \$left

Usage: **\$left(text,number)**

Category: text

Description:

Returns the first number characters from text. If number is less than 0, then the value used is the number of characters in text plus number (e.g.: `$left(abcd,-1)` is the same as `$left(abcd,3)`). If number 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"
$left(ABC,-2) ==> "A"
$left(ABC,-4) ==> ""
```

9.2.8 \$len

Usage: **\$len(text)**

Category: text

Description:

Returns the number of characters in text.

Example:

The following statements will return the values indicated:

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

$set(foo,ABC)
$len(%foo%) ==> "3"

$len()      ==> "0"
$len(ABC)   ==> "3"
```

9.2.9 \$lower

Usage: **\$lower(text)**

Category: text

Implemented: Picard

Description:

Returns text in lower case.

Example:

The following statement will return the value indicated:

```
$title(tHe houR is upOn uS) ==> "the hour is upon us"
```

9.2.10 \$num

Usage: **\$num(number,length)**

Category: text

Description:

Returns number formatted to length digits, where number and length are integers and length 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) ==> "000000000000000000123"
$num(123,50) ==> "000000000000000000123"
$num(123,5.5) ==> ""
$num(1.23,10) ==> "0000000000"
```

9.2.11 \$pad

Usage: **\$pad(text,length,character)**

Category: text

Description:

Pads the text to the length provided by adding as many copies of character as needed to the beginning of the string. For the padded length to be correct, character 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,+) ==> ""

```

9.2.12 \$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_"

```

9.2.13 \$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"
```

9.2.14 \$right

Usage: **\$right(text,number)**

Category: text

Description:

Returns the last number characters from text. If number is less than 1, then the value used is the number of characters in text plus number (e.g.: `$right(abcd,0)` is the same as `$right(abcd,4)`). If number 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) ==> ""
```

9.2.15 \$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](#).

Note: When entering regular expressions into Picard scripts you have to escape a backslash “\”, dollar sign “\$”, comma “,” and the left and right parentheses “(” and “)” in order to force Picard to not interpret them as part of the script command. This is done by inserting a backslash before the character to be escaped. For example, the regular expression `^\s*([0-9,\.\]*)$` would have to be entered as `^\s*\([0-9\,\.\]*\)\$`.

Example:

The following statements will return the values indicated:

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

9.2.16 \$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](#).

If a marked subexpression is defined using parentheses within the search pattern, only the pattern captured by the subexpression will be returned. If more than one marked subexpression is defined and matched, only the pattern captured by the first subexpression will be returned. If more than one marked subexpression is defined and not all are matched, an empty string will be returned. If no subexpression is specified, then the pattern captured by the whole search expression will be returned.

Note: When entering regular expressions into Picard scripts you have to escape a backslash “\”, dollar sign “\$”, comma “,” and the left and right parentheses “(” and “)”

“)” in order to force Picard to not interpret them as part of the script command. This is done by inserting a backslash before the character to be escaped. For example, the regular expression `^\s*([0-9,\.]*)$` would have to be entered as `^\s*\([0-9\,\.\.]*\)\$`.

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) ==> ""
$rsearch(test,t) ==> "t"
$rsearch(test,s) ==> "s"
$rsearch(test,\(e\).*s) ==> "e"
$rsearch(test,\(e\).*\s) ==> "e"
$rsearch(test,\(e\).*x) ==> ""
$rsearch(test,\(e\).*\s(x)) ==> ""
```

9.2.17 \$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.)
```

9.2.18 \$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 index is left blank, it will default to the start of the string. If the end index is left blank or not included, it will default to the end of the string. 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)	==>	" "

9.2.19 \$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:

<code>\$swapprefix(%albumartist%)</code>	<code>==></code>	<code>"Le Butcherettes"</code>
<code>\$swapprefix(%albumartist%,le)</code>	<code>==></code>	<code>"Le Butcherettes"</code>
<code>\$swapprefix(%albumartist%,L)</code>	<code>==></code>	<code>"Le Butcherettes"</code>
<code>\$swapprefix(%albumartist%,A,An,The,Le)</code>	<code>==></code>	<code>"Butcherettes, Le"</code>

9.2.20 \$title

Usage: **\$title(text)**

Category: text

Implemented: Picard 2.1

Description:

Returns text with the first character in every word capitalized. Note that other characters in the words will not be modified, which allows the preservation of all upper-case acronyms such as “BBC”. To only have the first character of each word capitalized you could first change the text to lower-case.

Examples:

The following statements will return the values indicated:

```
$set(foo,tHe houR is upOn uS)
$title(%foo%)           ==> "The HouR Is UpOn US"
$title($lower(%foo%))  ==> "The Hour Is Upon Us"

$set(bar,THIS TEXT IS ALL CAPITALS)
$title(%bar%)           ==> "THIS TEXT IS ALL CAPITALS"
$title($lower(%bar%))  ==> "This Text Is All Capitals"

$set(baz,AC/DC recorded live at the BBC studio in London)
$title(%baz%)           ==> "AC/DC Recorded Live At The BBC Studio In_
↳ London"
$title($lower(%baz%))  ==> "Ac/Dc Recorded Live At The Bbc Studio In_
↳ London"
```

9.2.21 \$trim

Usage: **\$trim(text[,character])**

Category: text

Description:

Trims all leading and trailing whitespaces from text. The optional second parameter character specifies the character to trim. If multiple characters are provided in character, 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"
```

9.2.22 \$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) ==> ""
```

9.2.23 \$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"
```

9.3 Multi-Value Functions

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

9.3.1 \$cleanmulti

Usage: **\$cleanmulti(name)**

Category: multi-value

Implemented: Picard 2.8

Description:

Removes all empty elements from the multi-value variable name.

Example:

The following statements will return the values indicated:

```
$setmulti(test,One; ; Two; Three)
%test%                               ==>  "One; ; Two; Three"
$cleanmulti(test)
%test%                               ==>  "One; Two; Three"
```

9.3.2 \$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)  ==>  ""
```

9.3.3 \$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"
```

(continues on next page)

(continued from previous page)

<code>\$setmulti(bar,First:A; Second:B)</code>	
<code>\$join(%bar%, >>)</code>	<code>==> "First:A >> Second:B"</code>
<code>\$join(%bar%, >> ,:)</code>	<code>==> "First >> A; Second >> B"</code>
<code>\$join(First:A; Second:B,)</code>	<code>==> "First:ASecond:B"</code>
<code>\$join(First:A; Second:B, >>)</code>	<code>==> "First:A >> Second:B"</code>
<code>\$join(First:A; Second:B, >> ,:)</code>	<code>==> "First >> A; Second >> B"</code>

9.3.4 \$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:

<code>\$set(foo,)</code>	
<code>\$lenmulti(%foo%)</code>	<code>==> "0"</code>
<code>\$set(foo,A; B; C)</code>	
<code>\$lenmulti(%foo%)</code>	<code>==> "1"</code>
<code>\$setmulti(foo,A; B; C)</code>	
<code>\$lenmulti(%foo%)</code>	<code>==> "3"</code>
<code>\$lenmulti(A; B; C)</code>	<code>==> "3"</code>
<code>\$lenmulti(A:A; B:B; C:C,:)</code>	<code>==> "4"</code>
<code>\$lenmulti(,)</code>	<code>==> "0"</code>
<code>\$lenmulti(,:)</code>	<code>==> "0"</code>
<code>\$lenmulti(A; B; C,)</code>	<code>==> "1"</code>

9.3.5 \$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.

Empty elements are automatically removed from the output.

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"

$setmulti(baz,A; B; x; C)
$map(%baz%,$if($eq(%_loop_value%,x),,%_loop_count%=_loop_value%))
                                                    ==>  "1=A; 2=B; 4=C"
$map(First:A; Second:B,
```

(continues on next page)

(continued from previous page)

```
$upper(%_loop_count%=%_loop_value%)) ==> "1=FIRST:A; 2=SECOND:B"
↪ "
```

9.3.6 \$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 by default the pattern to be matched is case-sensitive and can appear anywhere in the tag.

As of version 2.7, you can explicitly define a regular expression in the form /pattern/flags. The only supported flag is “i” (ignore case). For more information about regular expressions, please see the [article on Wikipedia](#).

Note: When entering regular expressions into Picard scripts you have to escape a backslash “\”, dollar sign “\$”, comma “,” and the left and right parentheses “(” and “)” in order to force Picard to not interpret them as part of the script command. This is done by inserting a backslash before the character to be escaped. For example, the regular expression `^\s*([0-9,\.]*)$` would have to be entered as `^\s*\([0-9\,\.\]*\)\$`.

Example:

With the performer tags as performer:guitar = “Ann”, performer:rhythm-guitar = “Bob” and performer:drums (drum kit) = “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(/Guitar/i) ==> "Ann, Bob"
$performer(/Guitar/) ==> ""
$performer(/^guitar/) ==> "Ann"
$performer(/^Guitar/i) ==> "Ann"
$performer(drums \()) ==> "Cindy"
```

(continues on next page)

(continued from previous page)

```

$performer(\(drum kit\)) ==> "Cindy"
$performer()             ==> "Ann, Bob, Cindy"
$performer(, / )         ==> "Ann / Bob / Cindy"

```

9.3.7 \$replacemulti

Usage: **\$replacemulti(name,search,replace[,separator])**

Category: multi-value

Implemented: Picard 2.6.1

Description:

Replaces occurrences of search with replace in the multi-value variable name and returns the resulting multi-value variable string with the elements separated by separator (or the default separator of a semicolon followed by a space “; ” if not passed).

Empty elements are automatically removed from the output.

Example:

The following statements will return the values indicated:

```

$setmulti(foo,Electronic; Idm; Techno)
$replacemulti(%foo%,Idm,IDM)           ==> "Electronic; IDM;
↳ Techno"

$setmulti(foo,Electronic; Jungle; Bardcore)
$replacemulti(%foo%,Bardcore,Hardcore) ==> "Electronic; Jungle;
↳ Hardcore"

$setmulti(foo,One; Two; Three)
$replacemulti(%foo%,Four,Five)         ==> "One; Two; Three"

$setmulti(foo,Four; Five; Six)
$replacemulti(%foo%,Five,)             ==> "Four; Six"

```

9.3.8 `$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"
```

9.3.9 `$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"
```

9.3.10 \$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"
```

(continues on next page)

(continued from previous page)

```
$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(:,)              ==>  ""
```

9.3.11 \$unique

Usage: **\$unique(name[,case_sensitive[,separator]])**

Category: multi-value

Implemented: Picard 2.6.1

Description:

Returns a sorted copy of the multi-value variable name with duplicate elements removed. By default, the comparison ignores the case of the elements; however, this can be changed by setting `case_sensitive` to a non-empty value. 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 \$unique will return an empty string.

Note: When performing a (default) case-insensitive comparison, the last matching element will be used in the result. For example, if the multi-value variable contained ‘abc’, ‘Abc’, ‘ABc’ and ‘ABC’ in that order, then the element ‘ABC’ would be included in the output.

Example:

The following statements will return the values indicated:

```
$setmulti(foo,a; A; B; b; cd; Cd; cD; CD; a; A; b)
$set(bar,a; A; B; b; cd; Cd; cD; CD; a; A; b)

$unique(%foo%)      ==>  "A; CD; b"
$unique(%bar%)      ==>  "a; A; B; b; cd; Cd; cD; CD; a; A; b"
$unique(%foo%,1)    ==>  "A; B; CD; Cd; a; b; cd; cd"

$unique(a; A; B; b; cd; Cd; cD; CD; a; A; b) ==>  "A; CD; b"
```

9.4 Mathematical Functions

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

9.4.1 \$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)  ==> ""
```

9.4.2 \$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"
```

(continues on next page)

(continued from previous page)

```
$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,)     ==> ""
```

9.4.3 \$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.: $\text{\$mod}(x,y,z) = (x \% y) \% z$). 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)    ==> ""
```

9.4.4 \$mul

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

Category: mathematical

Description:

Multiplies x by y. Can be used with an arbitrary number of arguments (i.e.: $\text{\$mul}(x, y, z) = (x * y) * z$). 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)    ==> ""
```

9.4.5 \$sub

Usage: **\$sub(x,y,*args)**

Category: mathematical

Description:

Subtracts y from x. Can be used with an arbitrary number of arguments (i.e.: $\text{\$sub}(x, y, z) = (x - y) - z$). 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) ==> ""
```


9.5 Conditional Functions

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

Warning: Formatting the code in your scripts by adding things like spaces, tabs and newlines could affect the results of conditional tests because these characters are not ignored. For example,

```
$set(test,)
$if(
    %test%,
    $set(test1,Not Empty),
    $set(test1,Empty)
)
$if(%test%,$set(test2,Not Empty),$set(test2,Empty))
```

will return “Not Empty” for %test1%, but “Empty” for %test2%. The different values are a result of the indentation in the formatted code.

The conditional scripting functions are:

9.5.1 \$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:

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

9.5.2 \$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:

<code>\$endswith(The time is now,is now)</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>
<code>\$endswith(The time is now,is NOW)</code>	<code>==></code>	<code>" "</code>	<code>(False)</code>
<code>\$endswith(The time is now,)</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>
<code>\$endswith(,)</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>
<code>\$endswith(,now)</code>	<code>==></code>	<code>" "</code>	<code>(False)</code>

9.5.3 \$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:

<code>\$eq(,a)</code>	<code>==></code>	<code>" "</code>	<code>(False)</code>
<code>\$eq(a,)</code>	<code>==></code>	<code>" "</code>	<code>(False)</code>
<code>\$eq(a,A)</code>	<code>==></code>	<code>" "</code>	<code>(False)</code>
<code>\$eq(a,a)</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>

9.5.4 `$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:

<code>\$eq_all(A,A,B,C)</code>	<code>==></code>	<code>" "</code>	<code>(False)</code>
<code>\$eq_all(A,a,A,A)</code>	<code>==></code>	<code>" "</code>	<code>(False)</code>
<code>\$eq_all(A,A,A,A)</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>
<code>\$eq_all(,,,)</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>
<code>\$eq_all(,a,)</code>	<code>==></code>	<code>" "</code>	<code>(False)</code>

9.5.5 `$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:

<code>\$eq_any(A,A,B,C)</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>
<code>\$eq_any(A,a,A,A)</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>
<code>\$eq_any(A,a,b,c)</code>	<code>==></code>	<code>" "</code>	<code>(False)</code>
<code>\$eq_any(,,,)</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>
<code>\$eq_any(,a,b,c)</code>	<code>==></code>	<code>" "</code>	<code>(False)</code>

9.5.6 \$gt

Usage: **\$gt(x,y[,type])**

Category: conditional

Description:

Returns "1" (True) if x is greater than y using the comparison specified in type. Possible values of type are "int" (integer), "float" (floating point), "text" (case-sensitive text), "nocase" (case-insensitive text) and "auto" (automatically determine the type of arguments provided), with "auto" used as the default comparison method if type is not specified. The "auto" type will use the first type that applies to both arguments in the following order of preference: "int", "float" and "text".

Note: The type argument was added in Picard v2.9. Prior to that, if an argument was missing or was not an integer, the function would return an empty string.

Example:

The following statements will return the values indicated:

\$gt(0, -1)	==>	"1" (True)
\$gt(6,6)	==>	"" (False)
\$gt(6.6,6.5)	==>	"1" (True)
\$gt(b,a)	==>	"1" (True)
\$gt(B,a)	==>	"" (False)
\$gt(a,6)	==>	"1" (True)
\$gt(a,6.5)	==>	"1" (True)
\$gt(6,4,int)	==>	"1" (True)
\$gt(6.1,4,int)	==>	"" (False)
\$gt(a,6,int)	==>	"" (False)
\$gt(4.1,4,float)	==>	"1" (True)
\$gt(4.2,4.1,float)	==>	"1" (True)
\$gt(6,4,float)	==>	"1" (True)
\$gt(a,6.5,float)	==>	"" (False)
\$gt(2020-01-01,2020-01,text)	==>	"1" (True)
\$gt(abcd,abc,text)	==>	"1" (True)
\$gt(ac,abc,text)	==>	"1" (True)
\$gt(a,A,text)	==>	"1" (True)
\$gt(a,B,text)	==>	"1" (True)
\$gt(B,a,text)	==>	"" (False)
\$gt(B,a,nocase)	==>	"1" (True)

(continues on next page)

(continued from previous page)

<code>\$gt(b,A,nocase)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$gt(a,B,nocase)</code>	<code>==></code>	<code>"" (False)</code>
<code>\$gt(A,b,nocase)</code>	<code>==></code>	<code>"" (False)</code>

9.5.7 \$gte

Usage: **\$gte(x,y[,type])**

Category: conditional

Description:

Returns "1" (True) if x is greater than or equal to y using the comparison specified in type. Possible values of type are "int" (integer), "float" (floating point), "text" (case-sensitive text), "nocase" (case-insensitive text) and "auto" (automatically determine the type of arguments provided), with "auto" used as the default comparison method if type is not specified. The "auto" type will use the first type that applies to both arguments in the following order of preference: "int", "float" and "text".

Note: The type argument was added in Picard v2.9. Prior to that, if an argument was missing or was not an integer, the function would return an empty string.

Example:

The following statements will return the values indicated:

<code>\$gte(0,-1)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$gte(6,6)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$gte(6.6,6.5)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$gte(b,a)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$gte(B,a)</code>	<code>==></code>	<code>"" (False)</code>
<code>\$gte(a,6)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$gte(a,6.5)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$gte(6,4,int)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$gte(6.1,4,int)</code>	<code>==></code>	<code>"" (False)</code>
<code>\$gte(a,6,int)</code>	<code>==></code>	<code>"" (False)</code>
<code>\$gte(4.1,4,float)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$gte(4.2,4.1,float)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$gte(6,4,float)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$gte(a,6.5,float)</code>	<code>==></code>	<code>"" (False)</code>
<code>\$gte(2020-01-02,2020-01,text)</code>	<code>==></code>	<code>"1" (True)</code>

(continues on next page)

(continued from previous page)

<code>\$gte(abcd,abc,text)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$gte(ac,abc,text)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$gte(A,a,text)</code>	<code>==></code>	<code>"" (False)</code>
<code>\$gte(a,B,text)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$gte(B,a,text)</code>	<code>==></code>	<code>"" (False)</code>
<code>\$gte(A,a,nocase)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$gte(B,a,nocase)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$gte(b,A,nocase)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$gte(a,B,nocase)</code>	<code>==></code>	<code>"" (False)</code>
<code>\$gte(A,b,nocase)</code>	<code>==></code>	<code>"" (False)</code>

9.5.8 \$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:

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

9.5.9 \$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%,,,) ==> ""
```

9.5.10 \$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)
```

9.5.11 \$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)

```

9.5.12 \$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)

```


9.5.13 \$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)
```

9.5.14 \$is_multi

Usage: **\$is_multi(x)**

Category: conditional

Implemented: Picard 2.7

Description:

Returns true, if the argument is a multi-value tag and there is more than one element.

Example:

The following statements will return the values indicated:

```
$set(foo,a; b; c)
$is_multi(%foo%)      ==> "" (False)

$set(bar,)
$is_multi(%bar%)      ==> "" (False)

$setmulti(baz,a; b; c)
$is_multi(%baz%)      ==> "1" (True)

$is_multi(a; b; c)    ==> "1" (True)
$is_multi(a)          ==> "" (False)
```

9.5.15 \$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)
```

9.5.16 \$lt

Usage: **\$lt(x,y[,type])**

Category: conditional

Description:

Returns “1” (True) if x is less than y using the comparison specified in type. Possible values of type are “int” (integer), “float” (floating point), “text” (case-sensitive text), “nocase” (case-insensitive text) and “auto” (automatically determine the type of arguments provided), with “auto” used as the default comparison method if type is not specified. The “auto” type will use the first type that applies to both arguments in the following order of preference: “int”, “float” and “text”.

Note: The type argument was added in Picard v2.9. Prior to that, if an argument was missing or was not an integer, the function would return an empty string.

Example:

The following statements will return the values indicated:

```
$lt(-1,0)           ==> "1" (True)
$lt(6,6)            ==> "" (False)
$lt(6.5,6.6)        ==> "1" (True)
$lt(a,b)            ==> "1" (True)
$lt(6,a)            ==> "1" (True)
$lt(6.5,a)          ==> "1" (True)
```

(continues on next page)

(continued from previous page)

<code>\$lt(4,6,int)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lt(4,6.1,int)</code>	<code>==></code>	<code>"" (False)</code>
<code>\$lt(6,a,int)</code>	<code>==></code>	<code>"" (False)</code>
<code>\$lt(4,4.1,float)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lt(4.1,4.2,float)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lt(4,6,float)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lt(6.5,a,float)</code>	<code>==></code>	<code>"" (False)</code>
<code>\$lt(2020-01-01,2020-01-02,text)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lt(abc,abcd,text)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lt(abc,ac,text)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lt(A,a,text)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lt(B,a,text)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lt(a,A,text)</code>	<code>==></code>	<code>"" (False)</code>
<code>\$lt(a,B,nocase)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lt(A,b,nocase)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lt(B,a,nocase)</code>	<code>==></code>	<code>"" (False)</code>
<code>\$lt(b,A,nocase)</code>	<code>==></code>	<code>"" (False)</code>

9.5.17 \$lte

Usage: **\$lte(x,y[,type])**

Category: conditional

Description:

Returns “1” (True) if x is less than or equal to y using the comparison specified in type. Possible values of type are “int” (integer), “float” (floating point), “text” (case-sensitive text), “nocase” (case-insensitive text) and “auto” (automatically determine the type of arguments provided), with “auto” used as the default comparison method if type is not specified. The “auto” type will use the first type that applies to both arguments in the following order of preference: “int”, “float” and “text”.

Note: The type argument was added in Picard v2.9. Prior to that, if an argument was missing or was not an integer, the function would return an empty string.

Example:

The following statements will return the values indicated:

<code>\$lte(-1,0)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lte(6,6)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lte(6.5,6.6)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lte(a,b)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lte(6,a)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lte(6.5,a)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lte(4,6,int)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lte(4,6.1,int)</code>	<code>==></code>	<code>"" (False)</code>
<code>\$lte(6,a,int)</code>	<code>==></code>	<code>"" (False)</code>
<code>\$lte(4,4.1,float)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lte(4.1,4.2,float)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lte(4,6,float)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lte(6.5,a,float)</code>	<code>==></code>	<code>"" (False)</code>
<code>\$lte(2020-01-01,2020-02,text)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lte(abc,abcd,text)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lte(abc,ac,text)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lte(A,a,text)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lte(B,a,text)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lte(a,A,text)</code>	<code>==></code>	<code>"" (False)</code>
<code>\$lte(a,B,nocase)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lte(A,b,nocase)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$lte(B,a,nocase)</code>	<code>==></code>	<code>"" (False)</code>
<code>\$lte(b,A,nocase)</code>	<code>==></code>	<code>"" (False)</code>

9.5.18 \$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:

<code>\$ne(,a)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$ne(a,)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$ne(a,A)</code>	<code>==></code>	<code>"1" (True)</code>
<code>\$ne(a,a)</code>	<code>==></code>	<code>"" (False)</code>

9.5.19 `$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:

<code>\$ne_all(A,A,B,C)</code>	<code>==></code>	<code>" "</code>	<code>(False)</code>
<code>\$ne_all(A,a,A,A)</code>	<code>==></code>	<code>" "</code>	<code>(False)</code>
<code>\$ne_all(A,a,a,a)</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>
<code>\$ne_all(,,,)</code>	<code>==></code>	<code>" "</code>	<code>(False)</code>
<code>\$ne_all(,a,a)</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>

9.5.20 `$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:

<code>\$ne_any(A,A,B,C)</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>
<code>\$ne_any(A,a,A,A)</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>
<code>\$ne_any(A,A,A,A)</code>	<code>==></code>	<code>" "</code>	<code>(False)</code>
<code>\$ne_any(,,,)</code>	<code>==></code>	<code>" "</code>	<code>(False)</code>
<code>\$ne_any(,a,,)</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>

9.5.21 \$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
```

9.5.22 \$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)
```

9.5.23 \$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:

<code>\$startswith(The time is now.,The time)</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>
<code>\$startswith(The time is now.,The TIME)</code>	<code>==></code>	<code>""</code>	<code>(False)</code>
<code>\$startswith(The time is now.,)</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>
<code>\$startswith(,The)</code>	<code>==></code>	<code>""</code>	<code>(False)</code>
<code>\$startswith(,)</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>

9.6 Information Functions

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

9.6.1 \$countryname

Usage: **\$countryname(country_code[,translate])**

Category: text

Implemented: Picard 2.7

Description:

Returns the name of the country for the specified country code. If the country code is invalid an empty string will be returned. If `translate` is not blank, the output will be translated into the current locale language, otherwise it will be in English.

Examples:

Assuming that the user's locale has been set to Russian, the following statements will return the values indicated:

<code>\$set(foo,ca)</code>	
<code>\$countryname(%foo%)</code>	<code>==> "Canada"</code>
<code>\$countryname(%foo%,yes)</code>	<code>==> "Канада"</code>

(continues on next page)

(continued from previous page)

<code>\$countryname(ca)</code>	<code>==></code>	<code>"Canada"</code>
<code>\$countryname(ca,)</code>	<code>==></code>	<code>"Canada"</code>
<code>\$countryname(ca,)</code>	<code>==></code>	<code>"Канада"</code>
<code>\$countryname(ca,yes)</code>	<code>==></code>	<code>"Канада"</code>
<code>\$countryname(INVALID)</code>	<code>==></code>	<code>" "</code>
<code>\$countryname(INVALID,yes)</code>	<code>==></code>	<code>" "</code>

9.6.2 \$dateformat

Usage: **\$dateformat(date,[format],[date order])**

Category: information

Implemented: Picard 2.7

Description:

Returns the input date in the specified format, which is based on the standard Python `strftime` [format codes](#). If no format is specified the date will be returned in the form '2020-02-15' (year, month, day).

The "year", "month" and "day" portions of the date must be entered as numbers, and can be separated by any non-numeric characters. The default order for the input date is "ymd" (year, month, day). This can be changed by specifying a `date order`.

Valid entries for `date order` are:

- **ymd** - year, month, day (This is the default order.)
- **dmy** - day, month, year
- **mdy** - month, day, year

If either the date or format are invalid an empty string will be returned. If an invalid `date order` is specified, the default order "ymd" will be used.

Note: 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 equivalents %#d and %#m on Windows); element length specifiers (e.g.: %3Y); and hanging '%' at the end of the format string.

Examples:

The following statements will return the values indicated:

```
$set(foo,07.21.2021)
$set(bar,mdy)
$set(format,\%Y.\%m.\%d)
$dateformat(%foo%,%format%,%bar%)    ==>  "2021.07.21"

$dateformat(2021 07 21)               ==>  "2021-07-21"
$dateformat(2021.07.21)              ==>  "2021-07-21"
$dateformat(2021-07-21)              ==>  "2021-07-21"
$dateformat(2021-7-21)               ==>  "2021-07-21"
$dateformat(2021-7-21,\%B \%d\, \%Y) ==>  "July 21, 2021"

$dateformat(2021-07-21,,ymd)          ==>  "2021-07-21"
$dateformat(2021-07-21,,dmy)         ==>  ""
$dateformat(2021-07-21,,mdy)         ==>  ""
$dateformat(2021-July-21)             ==>  ""
$dateformat(2021)                    ==>  ""
$dateformat(2021-07)                 ==>  ""
$dateformat(,)                       ==>  ""
```

9.6.3 \$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](#). If no format is specified the date and time will be returned in the form '2020-02-15 14:26:32'.

Note: 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 equivalents %#d and %#m on Windows); element length specifiers (e.g.: %3Y); and hanging '%' at the end of the format string.

Examples:

The following statements will return the values indicated:

<code>\$datetime()</code>	<code>==></code>	<code>"2020-02-15 14:26:32"</code>
<code>\$datetime(\%Y-\%m-\%d \%H:\%M:\%S)</code>	<code>==></code>	<code>"2020-02-15 14:26:32"</code>
<code>\$datetime(\%Y-\%m-\%d)</code>	<code>==></code>	<code>"2020-02-15"</code>
<code>\$datetime(\%H:\%M:\%S)</code>	<code>==></code>	<code>"14:26:32"</code>
<code>\$datetime(\%B \%d, \%Y)</code>	<code>==></code>	<code>"February 15, 2020"</code>

9.6.4 \$day

Usage: **\$day(date[,date order])**

Category: information

Implemented: Picard 2.7

Description:

Returns the “day” portion of the input date.

The “year”, “month” and “day” portions of the date must be entered as numbers, and can be separated by any non-numeric characters. The default order for the input date is “ymd” (year, month, day). This can be changed by specifying a `date order`.

Valid entries for `date order` are:

- **ymd** - year, month, day (This is the default order.)
- **dmy** - day, month, year
- **mdy** - month, day, year

If the date is invalid an empty string will be returned. If an invalid `date order` is specified, the default order “ymd” will be used.

Examples:

The following statements will return the values indicated:

<code>\$set(foo,07.21.2020)</code>	
<code>\$set(bar,mdy)</code>	
<code>\$day(%foo%,%bar%)</code>	<code>==> "21"</code>
<code>\$day(2020 07 21)</code>	<code>==> "21"</code>
<code>\$day(2020.07.21)</code>	<code>==> "21"</code>
<code>\$day(2020-07-21)</code>	<code>==> "21"</code>
<code>\$day(2020-07-2)</code>	<code>==> "2"</code>
<code>\$noop(Invalid date order)</code>	
<code>\$day(2020-07-21,dym)</code>	<code>==> "21"</code>

(continues on next page)

(continued from previous page)

<code>\$day(,)</code>	<code>==></code>	<code>" "</code>
<code>\$day(-07 -2020 , dmy)</code>	<code>==></code>	<code>" "</code>

9.6.5 `$matchedtracks`

Usage: `$matchedtracks()`

Category: information

Implemented: Picard 0.12

Description:

Returns the number of matched tracks within a release.

Note: This function only works in File Naming scripts.

Example:

The following statements will return the values indicated:

<code>\$matchedtracks()</code>	<code>==></code>	<code>"3"</code> (if three of the tracks were matched)
--------------------------------	---------------------	--

9.6.6 `$max`

Usage: `$max(type,x,...)`

Category: information

Implemented: Picard 2.9

Description:

Returns the maximum value using the comparison specified in type.

Possible values of type are "int" (integer), "float" (floating point), "text" (case-sensitive text), "nocase" (case-insensitive text) and "auto" (automatically determine the type of arguments provided), with "auto" used as the default comparison method if type is not specified. The "auto" type will use the first type that applies to both arguments in the following order of preference: "int", "float" and "text".

Can be used with an arbitrary number of arguments. Multi-value arguments will be expanded automatically.

Example:

The following statements will return the values indicated:

<code>\$max(text,)</code>	<code>==> ""</code>
<code>\$max(text,,a)</code>	<code>==> "a"</code>
<code>\$max(text,a,)</code>	<code>==> "a"</code>
<code>\$max(text,abc)</code>	<code>==> "abc"</code>
<code>\$max(text,abc,abcd,ac)</code>	<code>==> "ac"</code>
<code>\$max(text,A,a)</code>	<code>==> "a"</code>
<code>\$max(text,a,B)</code>	<code>==> "a"</code>
<code>\$max(text,2020-01-01,2020-01-02,2020-02)</code>	<code>==> "2020-02"</code>
<code>\$max(int,1)</code>	<code>==> "1"</code>
<code>\$max(int,1,2)</code>	<code>==> "2"</code>
<code>\$max(int,1,2,3.1)</code>	<code>==> ""</code>
<code>\$max(int,1,2,a)</code>	<code>==> ""</code>
<code>\$max(int,1,2,)</code>	<code>==> ""</code>
<code>\$max(float,1)</code>	<code>==> "1.0"</code>
<code>\$max(float,1,2)</code>	<code>==> "2.0"</code>
<code>\$max(float,1,2,3.1)</code>	<code>==> "3.1"</code>
<code>\$max(float,2.1,2.11,2.111)</code>	<code>==> "2.111"</code>
<code>\$max(float,1,2,a)</code>	<code>==> ""</code>
<code>\$max(float,1,2,)</code>	<code>==> ""</code>
<code>\$max(nocase,a,B)</code>	<code>==> "B"</code>
<code>\$max(nocase,a,B,c)</code>	<code>==> "c"</code>
<code>\$setmulti(mv,x; y; z)</code>	
<code>\$max(text,%mv%)</code>	<code>==> "z"</code>
<code>\$max(text,a,%mv%)</code>	<code>==> "z"</code>
<code>\$max(text,x; y; z)</code>	<code>==> "z"</code>
<code>\$max(int,5,4; 6; 3)</code>	<code>==> "6"</code>
<code>\$max(float,5.9,4.2; 6; 3.35)</code>	<code>==> "6.0"</code>
<code>\$max(,1,2)</code>	<code>==> "2"</code>
<code>\$max(auto,1,2)</code>	<code>==> "2"</code>
<code>\$max(,1.1,2)</code>	<code>==> "2.0"</code>
<code>\$max(auto,1.1,2)</code>	<code>==> "2.0"</code>
<code>\$max(,1,2.1,a)</code>	<code>==> "a"</code>
<code>\$max(auto,1,2.1,a)</code>	<code>==> "a"</code>
<code>\$max(,a,A)</code>	<code>==> "a"</code>
<code>\$max(,a,B)</code>	<code>==> "a"</code>
<code>\$max(auto,a,A)</code>	<code>==> "a"</code>
<code>\$max(auto,a,B)</code>	<code>==> "a"</code>

9.6.7 \$min

Usage: **\$min(type,x,...)**

Category: information

Implemented: Picard 2.9

Description:

Returns the minimum value using the comparison specified in type.

Possible values of type are “int” (integer), “float” (floating point), “text” (case-sensitive text), “nocase” (case-insensitive text) and “auto” (automatically determine the type of arguments provided), with “auto” used as the default comparison method if type is not specified. The “auto” type will use the first type that applies to both arguments in the following order of preference: “int”, “float” and “text”.

Can be used with an arbitrary number of arguments. Multi-value arguments will be expanded automatically.

Example:

The following statements will return the values indicated:

<code>\$min(text,)</code>	<code>==> ""</code>
<code>\$min(text,,a)</code>	<code>==> ""</code>
<code>\$min(text,a,)</code>	<code>==> ""</code>
<code>\$min(text,abc)</code>	<code>==> "abc"</code>
<code>\$min(text,abc,abcd,ac)</code>	<code>==> "abc"</code>
<code>\$min(text,A,a)</code>	<code>==> "A"</code>
<code>\$min(text,a,B)</code>	<code>==> "B"</code>
<code>\$min(text,2020-01-01,2020-01-02,2020-02)</code>	<code>==> "2020-01-01"</code>
<code>\$min(int,1)</code>	<code>==> "1"</code>
<code>\$min(int,1,2)</code>	<code>==> "1"</code>
<code>\$min(int,1,2,3.1)</code>	<code>==> ""</code>
<code>\$min(int,1,2,a)</code>	<code>==> ""</code>
<code>\$min(int,1,2,)</code>	<code>==> ""</code>
<code>\$min(float,1)</code>	<code>==> "1.0"</code>
<code>\$min(float,1,2)</code>	<code>==> "1.0"</code>
<code>\$min(float,1.1,2,3)</code>	<code>==> "1.1"</code>
<code>\$min(float,2.1,2.11,2.111)</code>	<code>==> "2.1"</code>
<code>\$min(float,1,2,a)</code>	<code>==> ""</code>
<code>\$min(float,1,2,)</code>	<code>==> ""</code>
<code>\$min(nocase,a,B)</code>	<code>==> "a"</code>
<code>\$min(nocase,a,B,c)</code>	<code>==> "a"</code>

(continues on next page)

(continued from previous page)

<code>\$setmulti(mv,x; y; z)</code>	
<code>\$min(text,%mv%)</code>	<code>==> "x"</code>
<code>\$min(text,a,%mv%)</code>	<code>==> "a"</code>
<code>\$min(text,x; y; z)</code>	<code>==> "x"</code>
<code>\$min(int,5,4; 6; 3)</code>	<code>==> "3"</code>
<code>\$min(float,5.9,4.2; 6; 3.35)</code>	<code>==> "3.35"</code>
<code>\$min(,1,2)</code>	<code>==> "1"</code>
<code>\$min(auto,1,2)</code>	<code>==> "1"</code>
<code>\$min(,1,2.1)</code>	<code>==> "1.0"</code>
<code>\$min(auto,1,2.1)</code>	<code>==> "1.0"</code>
<code>\$min(,1,2.1,a)</code>	<code>==> "1"</code>
<code>\$min(auto,1,2.1,a)</code>	<code>==> "1"</code>
<code>\$min(,a,A)</code>	<code>==> "A"</code>
<code>\$min(,a,B)</code>	<code>==> "B"</code>
<code>\$min(auto,a,A)</code>	<code>==> "A"</code>
<code>\$min(auto,a,B)</code>	<code>==> "B"</code>

9.6.8 \$month

Usage: **\$month(date[,date order])**

Category: information

Implemented: Picard 2.7

Description:

Returns the “month” portion of the input date.

The “year”, “month” and “day” portions of the date must be entered as numbers, and can be separated by any non-numeric characters. The default order for the input date is “ymd” (year, month, day). This can be changed by specifying a `date order`.

Valid entries for `date order` are:

- **ymd** - year, month, day (This is the default order.)
- **dmy** - day, month, year
- **mdy** - month, day, year

If the date is invalid an empty string will be returned. If an invalid `date order` is specified, the default order “ymd” will be used.

Examples:

The following statements will return the values indicated:

```
$set(foo,07.21.2020)
$set(bar,mdy)
$month(%foo%,%bar%)      ==>  "07"

$month(2020 07 21)       ==>  "07"
$month(2020.07.21)       ==>  "07"
$month(2020-07-21)       ==>  "07"
$month(2020-7-21)        ==>  "7"

$noop( Invalid date order )
$month(2020-07-21,dym)   ==>  "07"

$month(,)                ==>  ""
$month(-21-2020,mdy)     ==>  ""
```

9.6.9 \$year

Usage: **\$year(date[,date order])**

Category: information

Implemented: Picard 2.7

Description:

Returns the “year” portion of the input date.

The “year”, “month” and “day” portions of the date must be entered as numbers, and can be separated by any non-numeric characters. The default order for the input date is “ymd” (year, month, day). This can be changed by specifying a date order.

Valid entries for date order are:

- **ymd** - year, month, day (This is the default order.)
- **dmy** - day, month, year
- **mdy** - month, day, year

If the date is invalid an empty string will be returned. If an invalid date order is specified, the default order “ymd” will be used.

Examples:

The following statements will return the values indicated:

```
$set(foo,07.21.2020)
$set(bar,mdy)
$year(%foo%,%bar%)      ==>  "2020"
```

(continues on next page)

(continued from previous page)

```
$year(2020 07 21)      ==> "2020"
$year(2020.07.21)      ==> "2020"
$year(2020-07-21)      ==> "2020"
$year(20-7-21)         ==> "20"

$noop( Invalid date order )
$year(2020-07-21,dym)  ==> "2020"

$year(,)               ==> ""
$year(07-21,mdy)       ==> ""
$year(21-07,dmy)       ==> ""

$noop( Month is not numeric )
$year(21-July-2020,dmy,1) ==> ""
```

9.7 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:

9.7.1 \$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%))
```

(continues on next page)

(continued from previous page)

```
$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%))
```

9.7.2 \$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...))
```

9.8 Miscellaneous Functions

The miscellaneous scripting functions are:

9.8.1 \$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:

<code>\$noop(A comment.)</code>	<code>==></code>	<code>""</code>
<code>\$noop(\$set(foo,Testing...))</code>	<code>==></code>	<code>""</code> (and "foo" is not set)

USING PICARD

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

10.1 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 existing in the files, or having the source CD available).

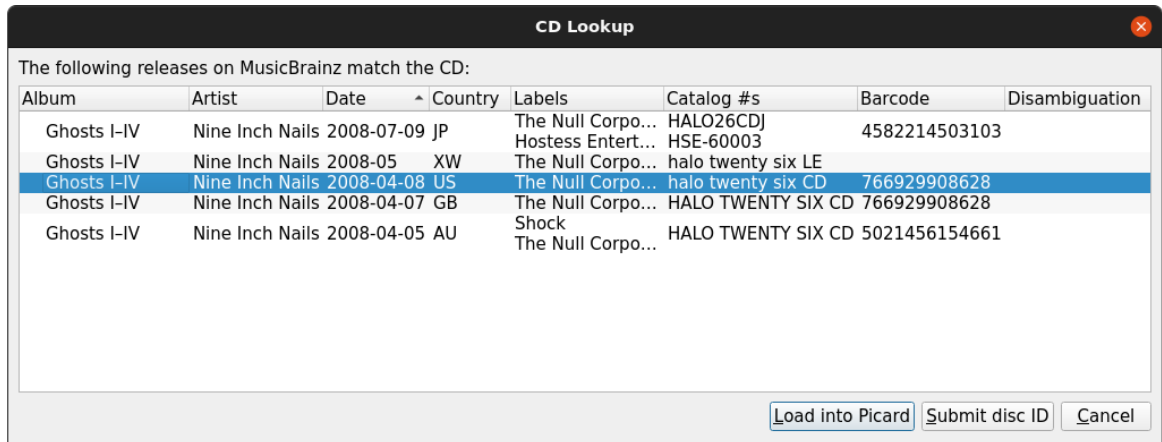
There are basically four main methods used to retrieve album information from the MusicBrainz database.

10.1.1 Lookup CD or Ripper Log

This is the preferred method of automatically identifying the album to retrieve, and should be used when you have the CD or [supported ripper log](#) 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.

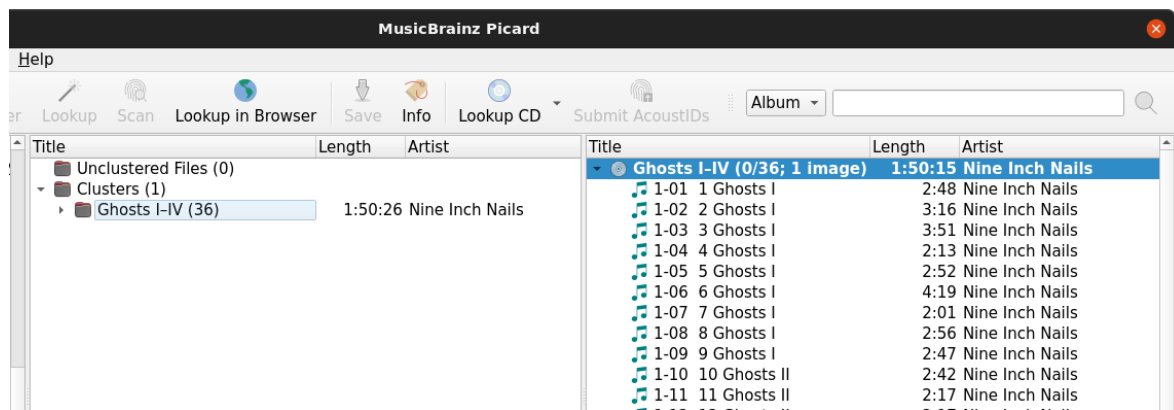
The steps to follow to lookup a CD are:

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. Alternatively, you can use a [supported ripper log file](#) to lookup the CD using the “*Tools → Lookup CD → From CD ripper log file...*” command. This will open a file browser dialog to allow you to select the log file to process. Either method will query the MusicBrainz database and display a list of matching releases.



2. Select the correct release from the list and click on the *Load into Picard* button. 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 *Submit disc ID* 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.

MusicBrainz outsidecontext ▾ My Data ▾

Search Artist ▾ Q

About Us ▾ Products ▾ Search ▾ Editing ▾ Documentation ▾ English ▾

Lookup CD

Matching CDs

We found discs matching the information you requested, listed below. If none of these are the release you are looking for, you may search using the form below in order to attach this disc to another MusicBrainz release.

Position	Title	Artist	Format	Country/Date	Label	Catalog#	Barcode	Tagger
1/2 (show tracklist)	Ghosts I-IV	Nine Inch Nails	CD	GB 2008-04-07	The Null Corporation	HALO TWENTY SIX CD	766929908628	
1/9 (show tracklist)	Ghosts I-IV	Nine Inch Nails	CD	XW 2008-05	The Null Corporation	halo twenty six LE		
1/3 (show tracklist)	Ghosts I-IV	Nine Inch Nails	CD	JP 2008-07-09	The Null Corporation, Hostess Entertainment Unlimited.	HALO26CDJ, HSE-60003	4582214503103	
1/2 (show tracklist)	Ghosts I-IV	Nine Inch Nails	CD	AU 2008-04-05	Shock (Australian independent), The Null Corporation	HALO TWENTY SIX CD	5021456154661	
1/2 (show tracklist)	Ghosts I-IV	Nine Inch Nails	CD	US 2008-04-08	The Null Corporation	halo twenty six CD	766929908628	

We used DiscID `tRNnyEChpaKYhU.TLG0gjNPDxcI` - to look up this information.

Search by artist

Artist:

Search

Search by release

Release title or MBID:

Search

Donate Wiki Forums IRC Bug Tracker Blog Twitter Use beta site Brought to you by [MetaBrainz Foundation](#) and our [sponsors](#) and [supporters](#). Cover Art provided by the [Cover Art Archive](#).

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

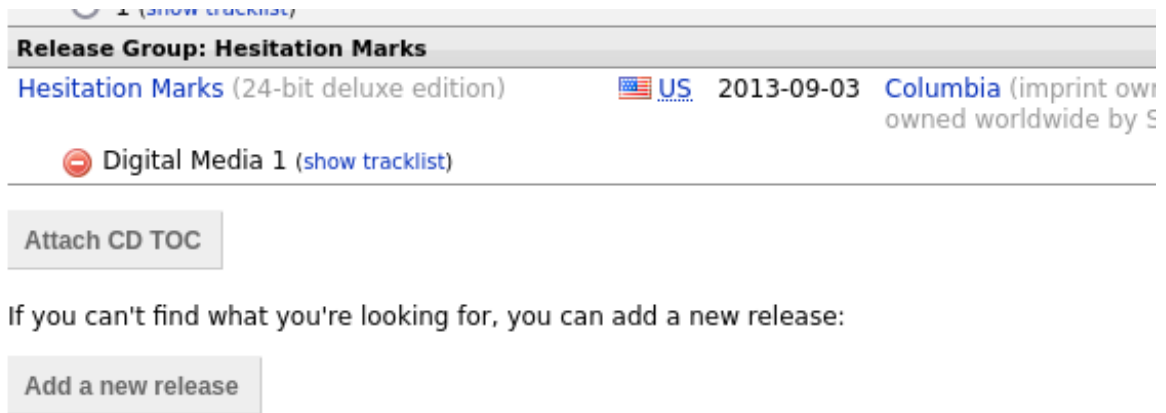
Attach CD TOC

You are viewing releases by [Nine Inch Nails](#).

Please select the medium you wish to attach this CD TOC to.

Release	Country/Date	Label	Catalog#	Barcode	Tagger
Release Group: Rusty Nails 3					
Rusty Nails 3					
Digital Media 1 (show tracklist)					
Release Group: 2005-06-12: Southside Festival, Germany					
2005-06-12: Southside Festival, Germany					
1 (show tracklist)					
Release Group: Ghosts I-IV					
Ghosts I-IV	AU 2008-04-05	Shock (Australian independent), The Null Corporation	HALO TWENTY SIX CD	5021456154661	
<input checked="" type="radio"/> CD 1: Ghosts I-II (show tracklist)					
<input type="radio"/> CD 2: Ghosts III-IV (show tracklist)					
Ghosts I-IV	GB 2008-04-07	The Null Corporation	HALO TWENTY SIX CD	766929908628	
<input type="radio"/> CD 1: Ghosts I-II (show tracklist)					
<input type="radio"/> CD 2: Ghosts III-IV (show tracklist)					

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

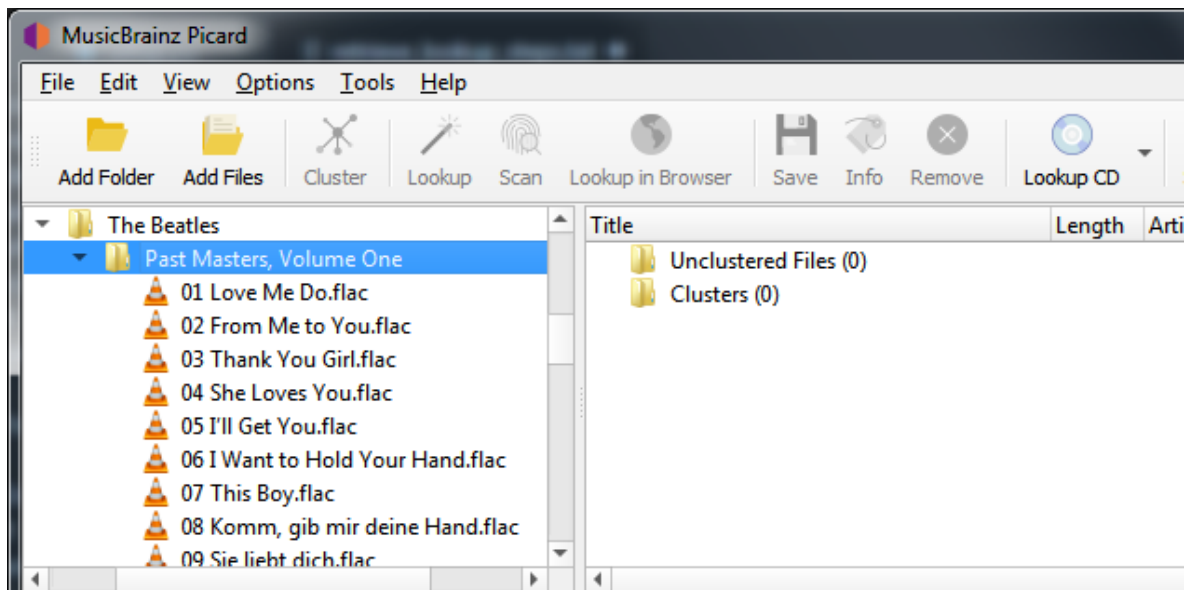


10.1.2 Lookup Files

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.

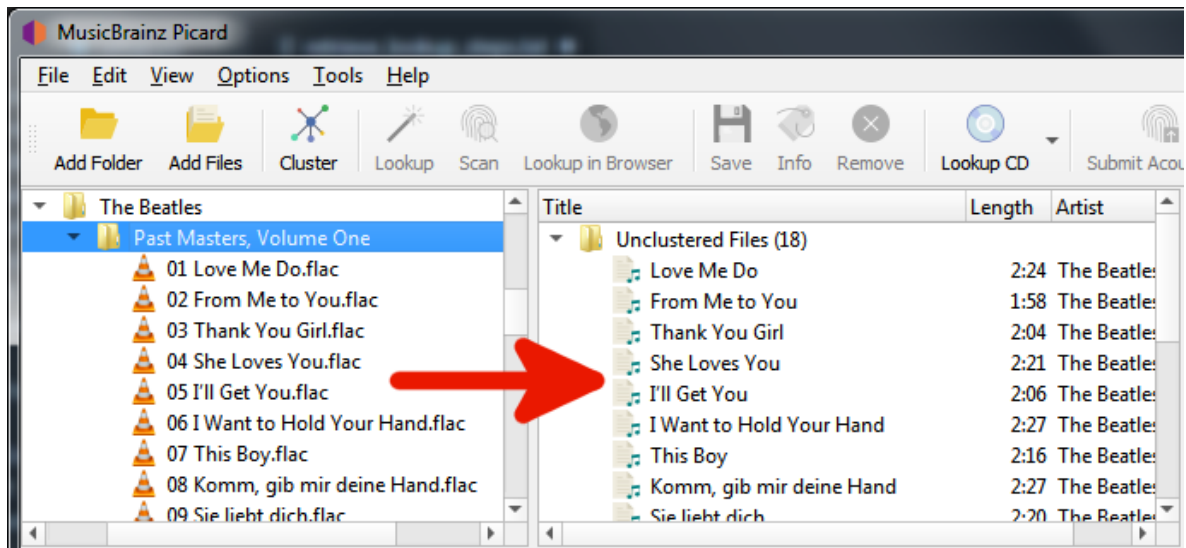
The steps to follow to lookup files are:

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".

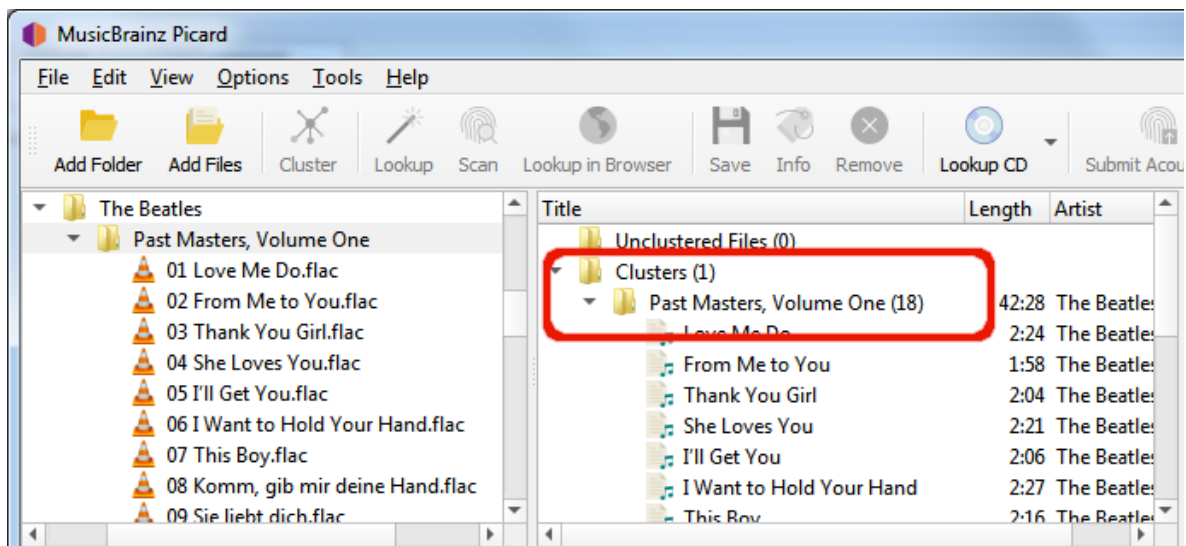


2. Drag the selected directory or files to the "Unclustered Files" folder, and wait for

Picard to process the files - the names will turn from grey to black.

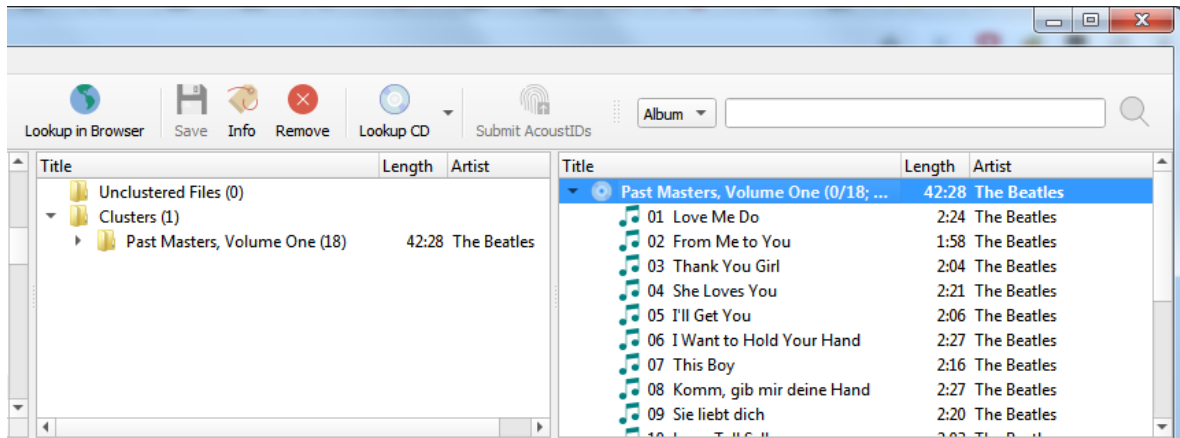


3. Use “Tools → Cluster” to group the files into album clusters.

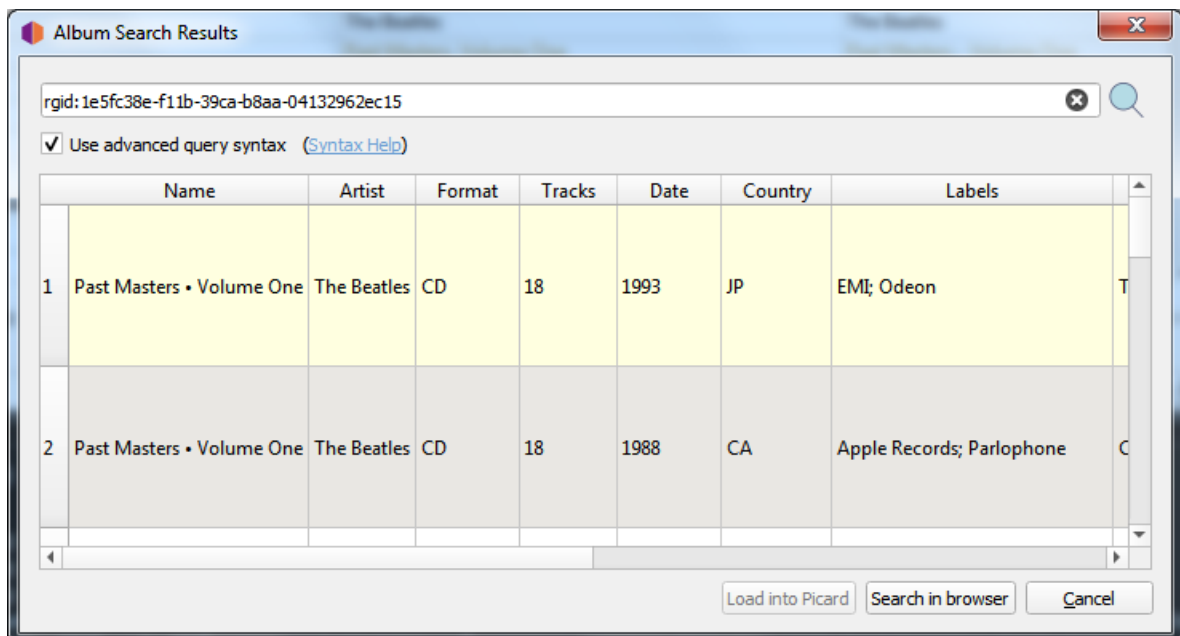


4. Select a clustered album and use “Tools → Lookup” to lookup the cluster. Depending on your previous metadata, the album will show up in the right-hand pane.

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 you're not sure that the album retrieved is correct, you can use *Tools → Show other album versions...* to open a window displaying all releases matched. From this window, you can select a different matching version to use, or refine the search criteria and perform a new search.



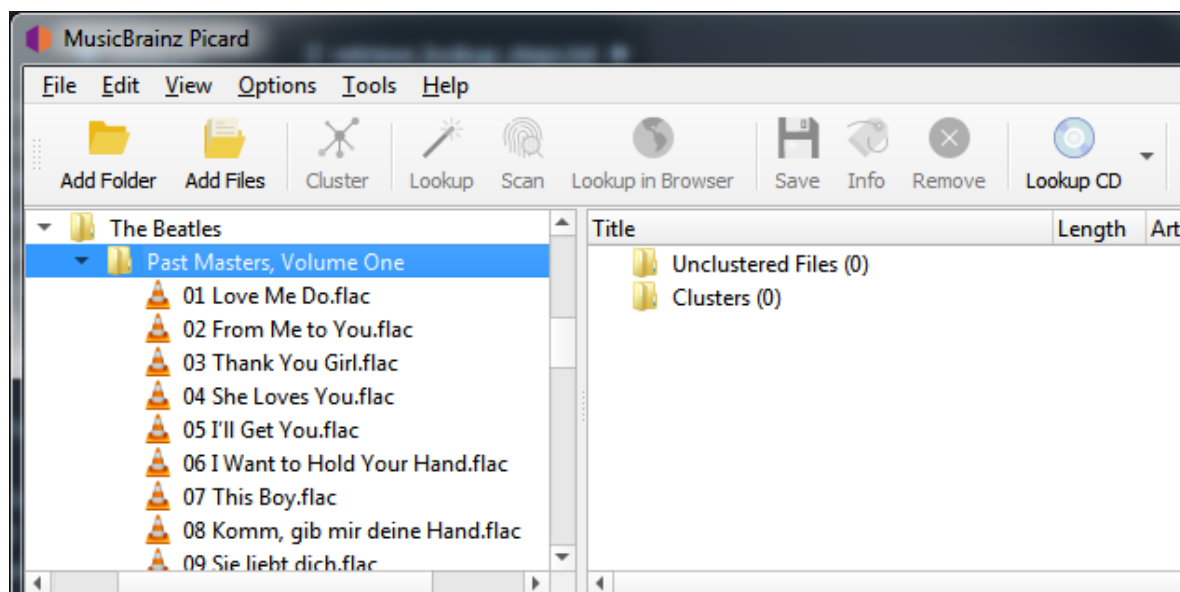
If no album was retrieved, or if the album retrieved was incorrect, you may have to try a different method such as scanning the files or a manual lookup.

10.1.3 Scan Files

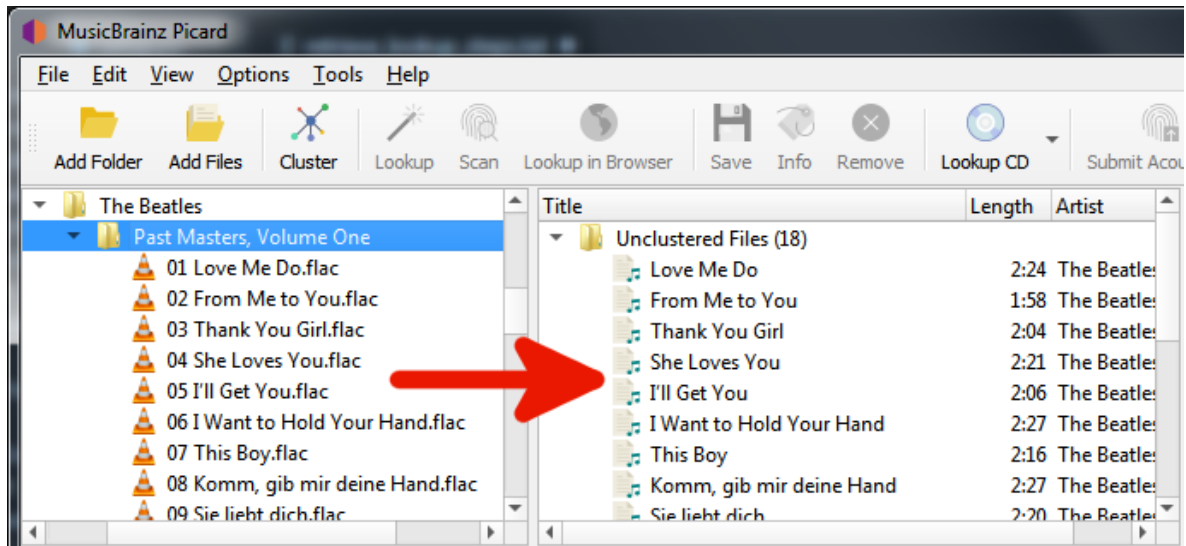
If your files are not grouped into albums and you don't have the CD available, this is the only remaining method of automatically identifying the album to retrieve. This is done by scanning the files to obtain their AcoustID fingerprints and then perform the lookup for the individual files by fingerprint. 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 scan and 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"*.

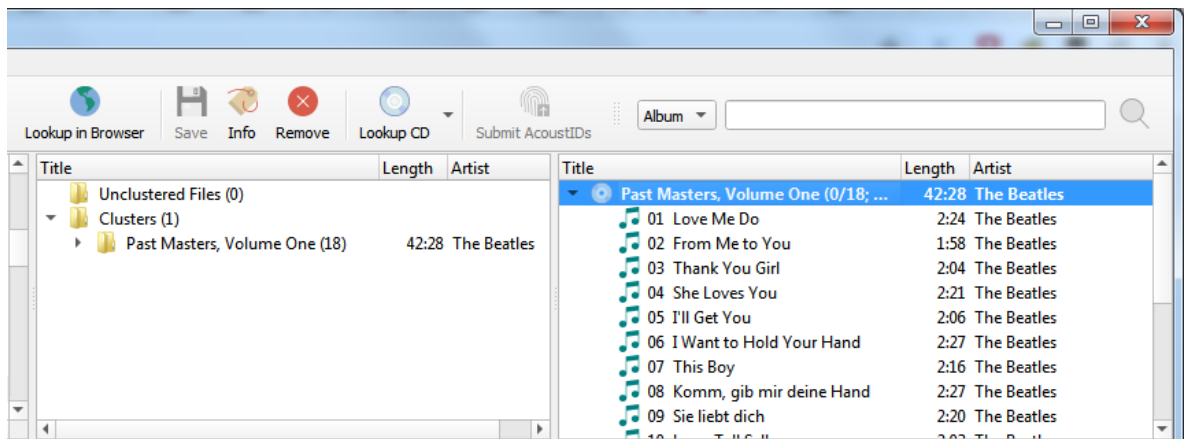


2. Drag the selected directory or files to the "Unclustered Files" folder, and wait for Picard to process the files - the names will turn from grey to black.



3. Select the desired files and use “Tools → Scan” to scan the files to determine their AcoustID fingerprints and lookup using this information. 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.

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.



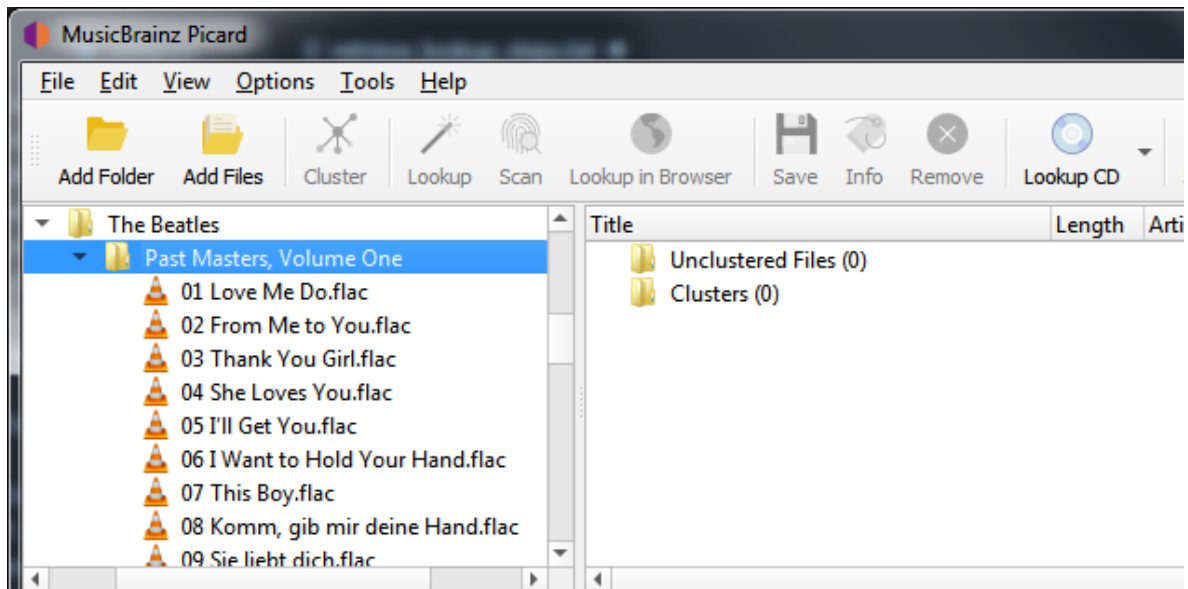
If no album was retrieved, or if the album retrieved was incorrect, you may have to try a different method such as clustering the files or a browser lookup.

10.1.4 Lookup in Browser

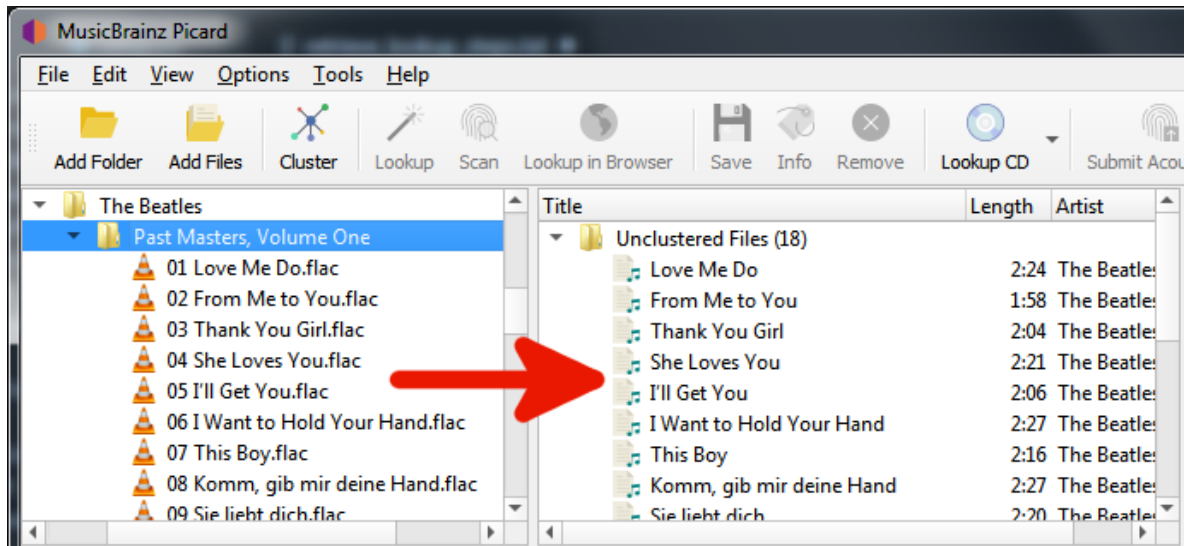
If none of the automated methods are available, or don't produce the desired results, you have the option of retrieving the album information by having Picard initiate a search on the MusicBrainz website using your web browser. There are two methods of initiating this search. The first method searches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

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

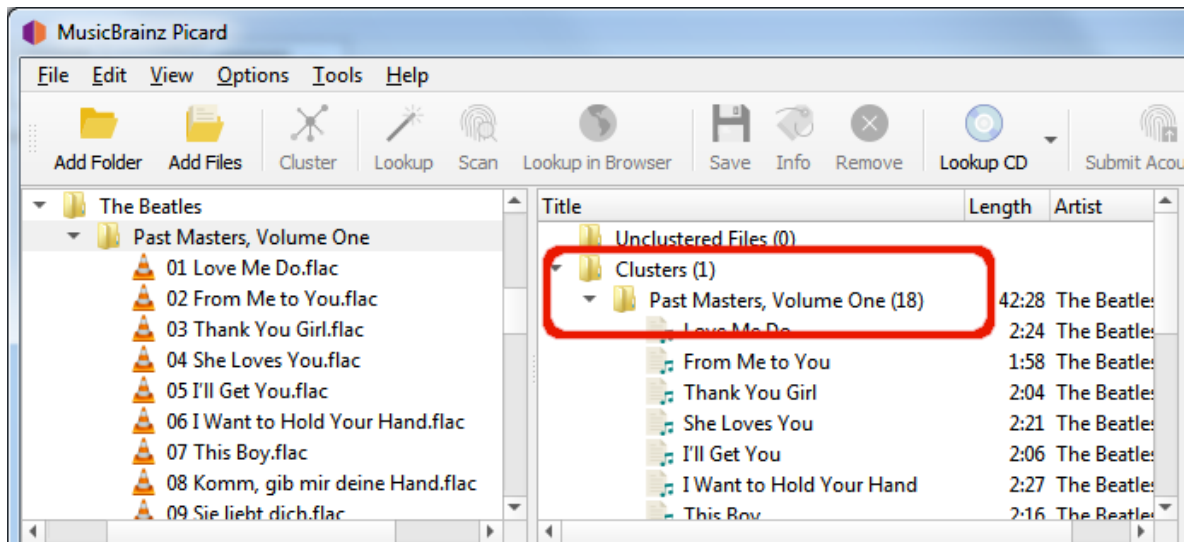
1. Add your files using *"Files → Add Files..."* or *"Files → Add Folder..."*. For ease of use it is recommended to use the internal File Browser to manage file system interactions. This is enabled from *"View → File Browser"*.



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.



- Use “Tools → Cluster” to group the files into album clusters if you want lookup a cluster.



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

MusicBrainz Tag Lookup Results

Found 2,321 results

Name	Artist	Format	Tracks	Country/Date	Label	Catalog#	Barcode	Language	Type	Status	Tagger
Past Masters, Volume One	The Beatles	CD	18	XE 1988	Parlophone	CD-BPM 1, CDP 7 90043 2	077779004324	eng / Latn	Album + Compilation	Official	
Past Masters, Volume One	The Beatles	CD	18	US 1988-03-08	Capitol Records, Parlophone	CDP 7 90043 2	077779004324	eng / Latn	Album + Compilation	Official	
Past Masters, Volume One	The Beatles	CD	18	CA 1988	Apple Records, Parlophone	C2 90043	077779004324	eng / Latn	Album + Compilation	Official	
Past Masters, Volume One	The Beatles	CD	18	JP -	EMI, Odeon	TOCP-8515, CP32-5601		eng / Latn	Album + Compilation	Official	
Past Masters, Volume One	The Beatles	CD	18	GB 1988-03-07	Parlophone	CD-BPM1, CDP 7 90043 2	077779004324	eng / Latn	Album + Compilation	Official	
Past Masters, Volume One	The Beatles	CD	18	JP 1988-03-11	EMI, Parlophone, Apple Records	TOCP-51125	4988006740082	eng / Latn	Album + Compilation	Official	
Past Masters, Volume One	The Beatles	CD	18	CA 1988	Parlophone	CDP 7 90043 2	077779004324	eng / Latn	Album + Compilation	Official	

- 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.

MusicBrainz Tag Lookup Results

Found 2,321 results

All You Need Is Love	The Beatles	(unknown)	2	US 1967-07-17				eng / Latn	Single	Official	
Let It Be	The Beatles	7" Vinyl	2	US 1970-03-11	Apple Records	2764	[none]	eng / Latn	Single	Official	
Free as a Bird	The Beatles	(unknown)	2	GB 1995-12-12				eng / Latn	Single	Official	
Nothing Is Real: The Making of "Strawberry Fields Forever"	The Beatles	(unknown)	14	DE 1992				eng / Latn	Other	Bootleg	

Search form:

Artist:

Release:

Track number:

Track:

Duration:

Filename:

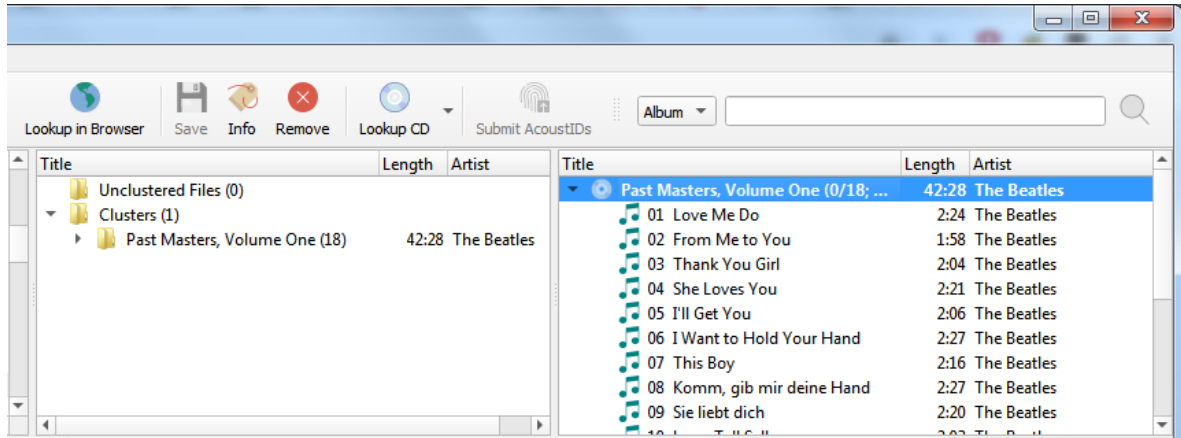
- Use the green arrow to load the information for a release into Picard.

MusicBrainz Tag Lookup Results

Found 2,321 results

Name	Artist	Format	Tracks	Country/Date	Label	Catalog#	Barcode	Language	Type	Status	Tagger
Past Masters, Volume One	The Beatles	CD	18	XE 1988	Parlophone	CD-BPM 1, CDP 7 90043 2	077779004324	eng / Latn	Album + Compilation	Official	
Past Masters, Volume One	The Beatles	CD	18	US 1988-03-08	Capitol Records, Parlophone	CDP 7 90043 2	077779004324	eng / Latn	Album + Compilation	Official	
Past Masters, Volume One	The Beatles	CD	18	CA 1988	Apple Records, Parlophone	C2 90043	077779004324	eng / Latn	Album + Compilation	Official	
Past Masters, Volume One	The Beatles	CD	18	JP -	EMI, Odeon	TOCP-8515, CP32-5601		eng / Latn	Album + Compilation	Official	
Past Masters, Volume One	The Beatles	CD	18	GB 1988-03-07	Parlophone	CD-BPM1, CDP 7 90043 2	077779004324	eng / Latn	Album + Compilation	Official	
Past Masters, Volume One	The Beatles	CD	18	JP 1988-03-11	EMI, Parlophone, Apple Records	TOCP-51125	4988006740082	eng / Latn	Album + Compilation	Official	
Past Masters, Volume One	The Beatles	CD	18	CA 1988	Parlophone	CDP 7 90043 2	077779004324	eng / Latn	Album + Compilation	Official	

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.

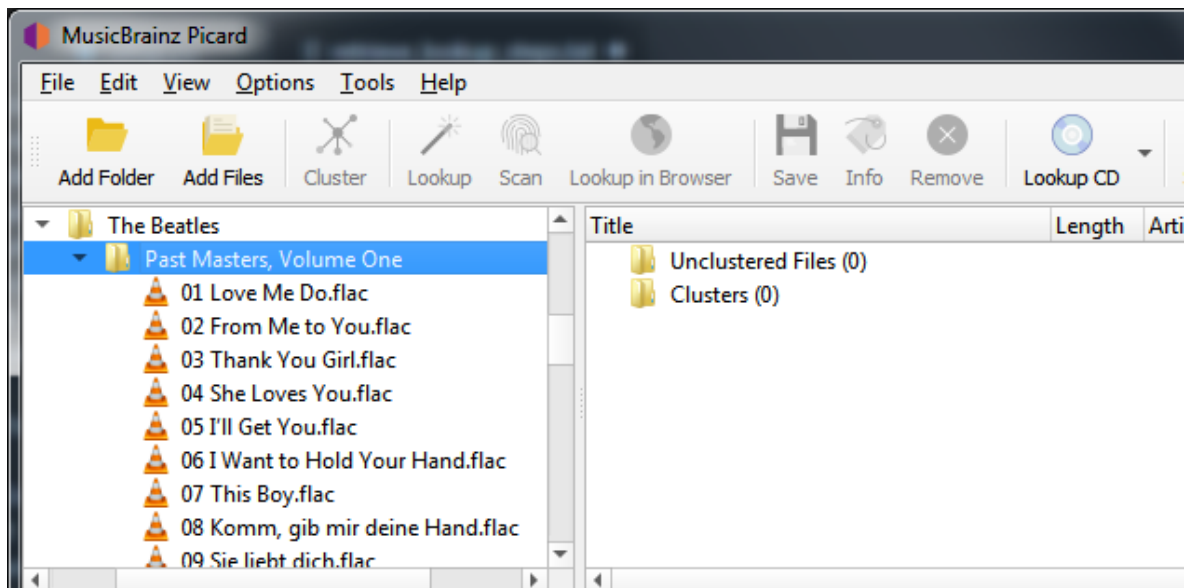


10.1.5 Manual Lookup

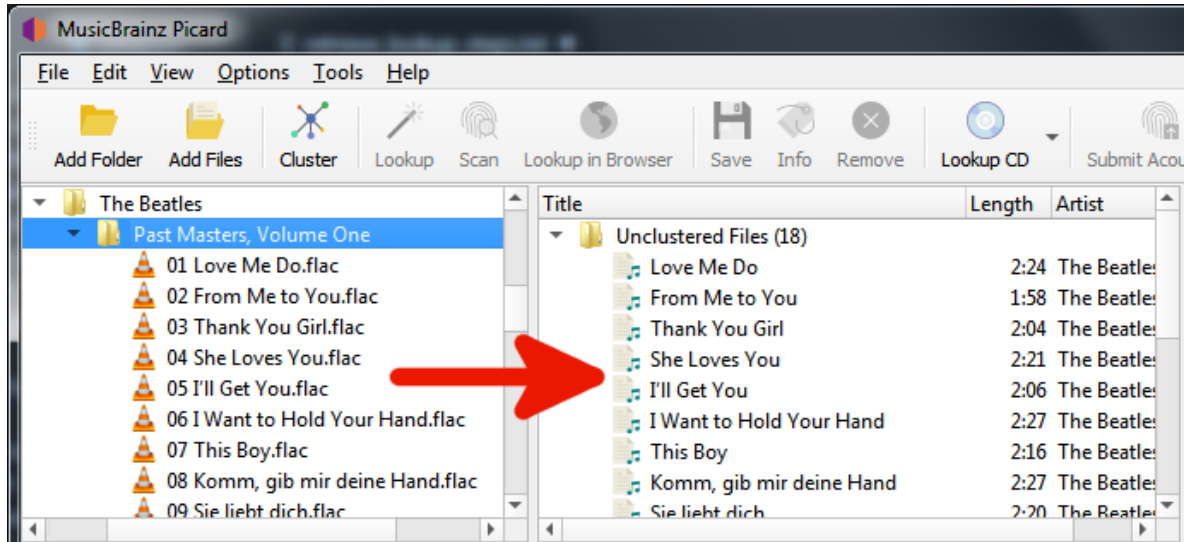
The second browser search method uses manually entered information as the search criterion.

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

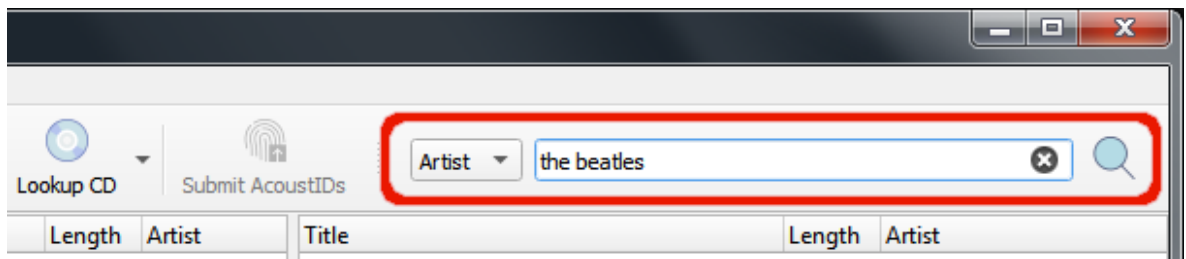
1. Add your files using “Files → Add Files...” or “Files → Add Folder...”. For ease of use it is recommended to use the internal File Browser to manage file system interactions. This is enabled from “View → File Browser”.



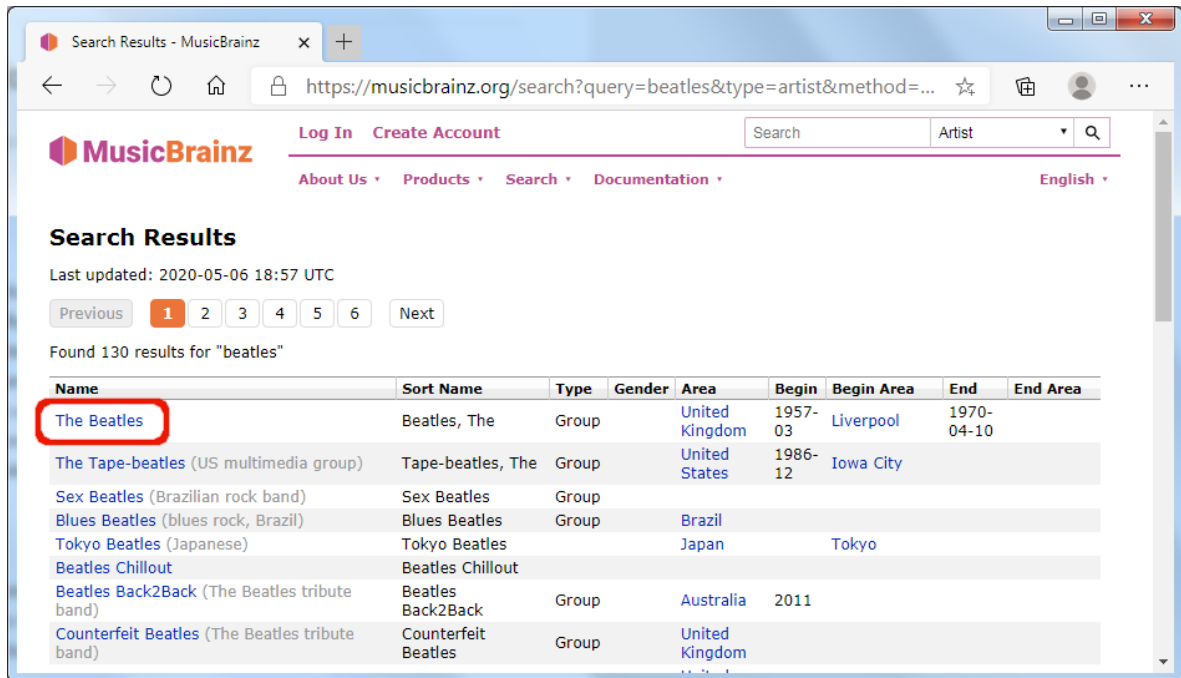
2. Drag the selected directory or files to the “Unclustered Files” folder, and wait for Picard to process the files - the names will turn from grey to black.




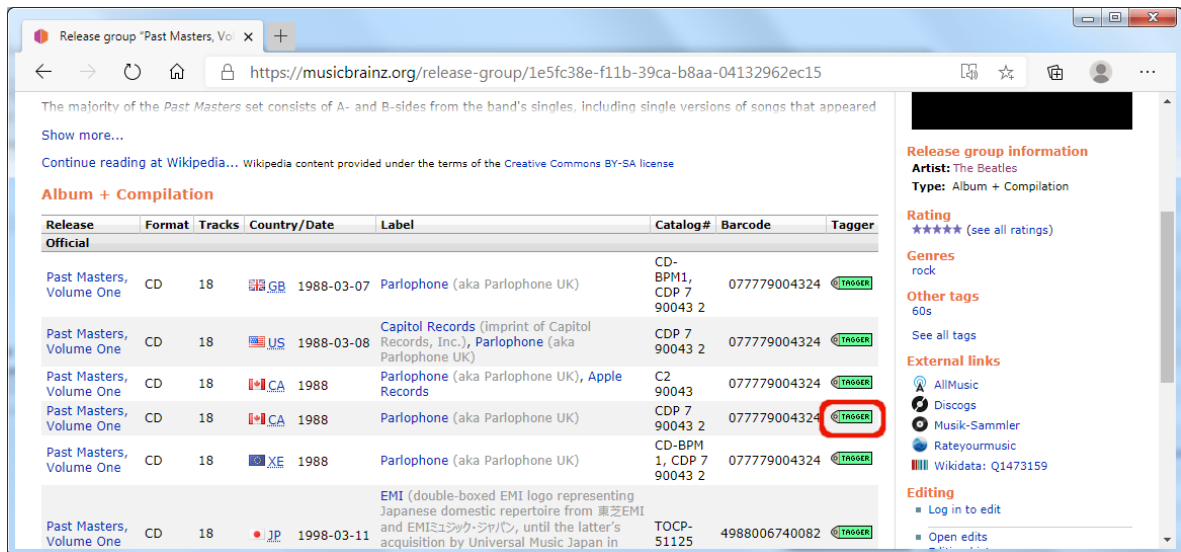
3. Enter your search information into the search box and select the type of records to search, then click the magnifying glass symbol to initiate the search. This will open the [MusicBrainz website](https://musicbrainz.org/) in your browser.



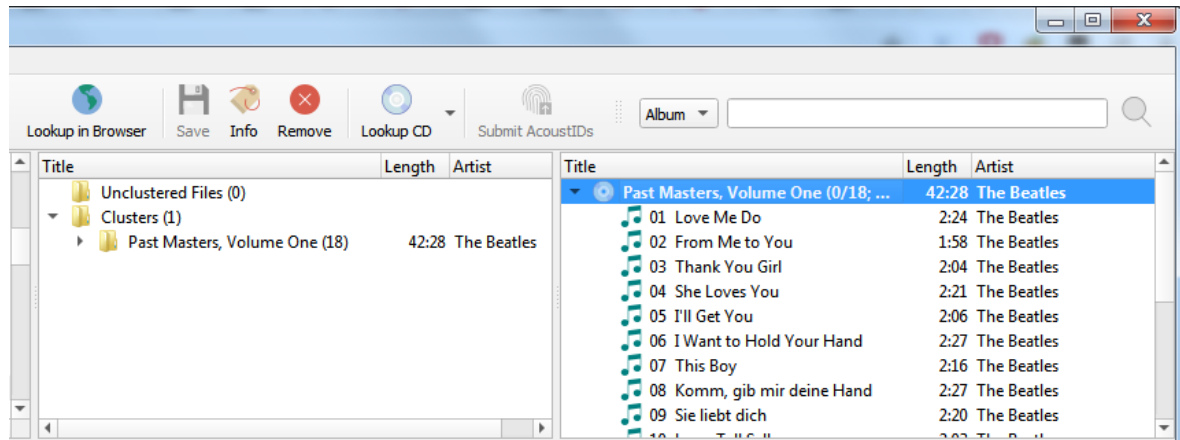
4. Continue to drill down by clicking on the appropriate links until you get to the release that you want to retrieve.



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.

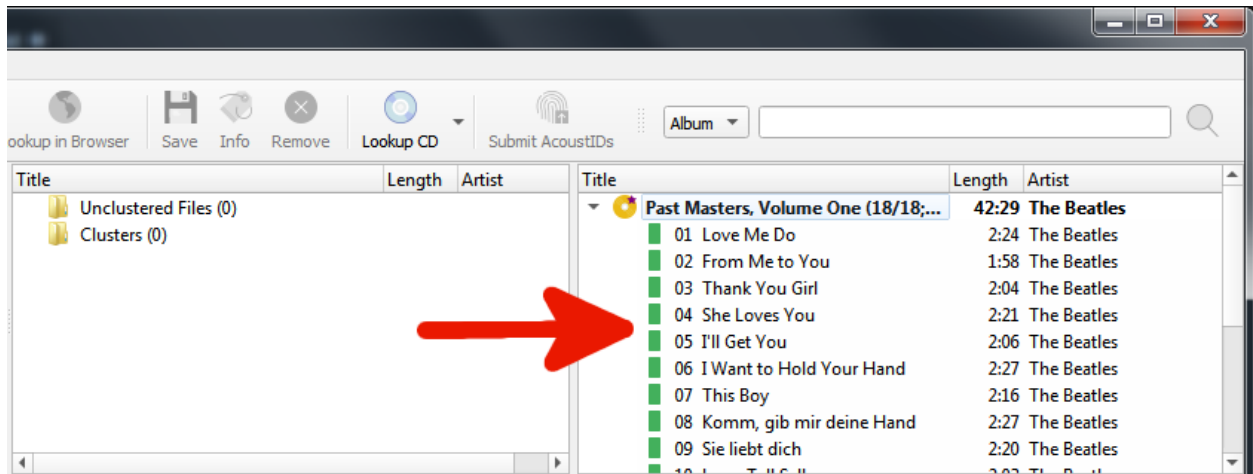


Note: If you enter a link to the desired entry (e.g.: <https://musicbrainz.org/release/9383a6f5-9607-4a36-9c68-8663aad3592b>) in the search box in Picard, the entry will be loaded directly without opening a browser window.

10.2 Matching Files to Tracks

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

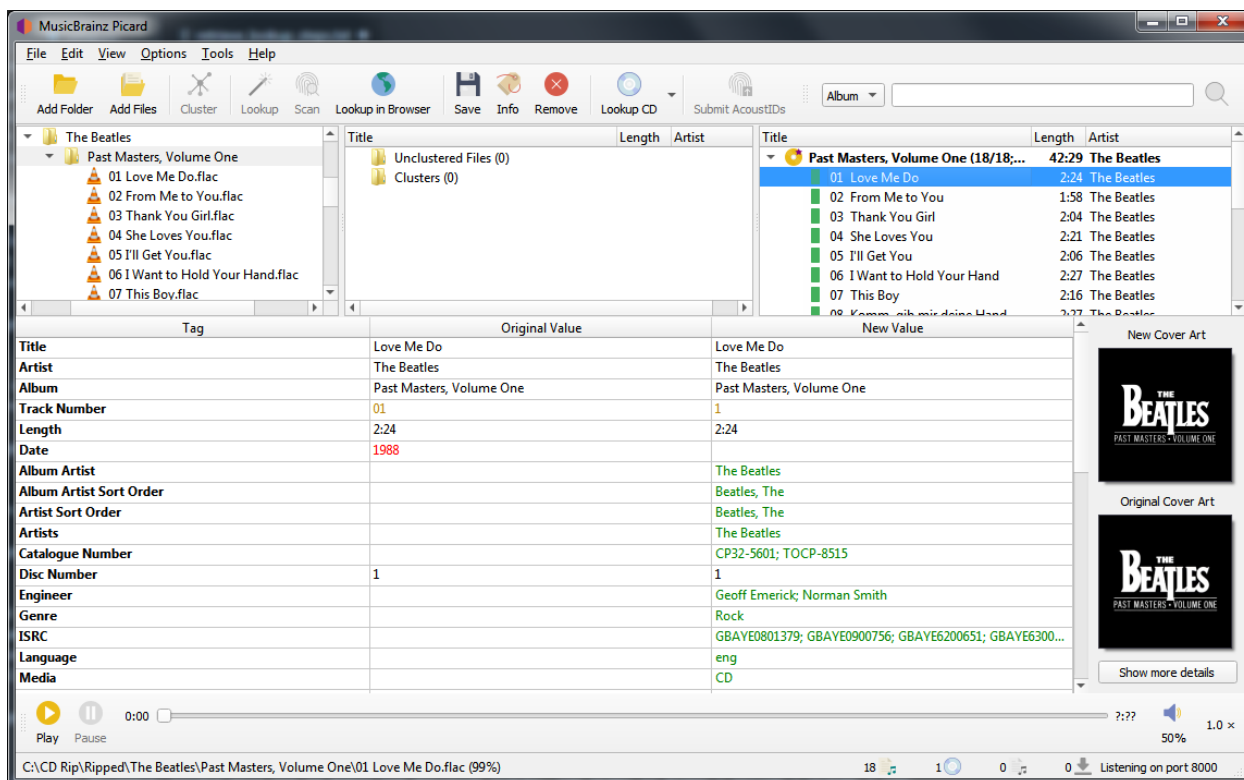
Once you have retrieved the desired album information into the right-hand pane, the next step is to match the files from the left-hand pane to the corresponding track in the right-hand pane. A music symbol in front of a track number in the right-hand pane indicates that there has been no file assigned 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.



Note: If you drag and drop multiple files onto a specific track the first selected file will be matched to the track on which you dropped the files. The rest of the selected files will be matched to the following tracks in order. This allows you to quickly match multiple files to a sequence of tracks. If you want to match all files to a single track instead you can hold the **Alt** key while dropping the files.

If you drop multiple files onto an album Picard will try to match the files to the tracks based on the metadata.

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.



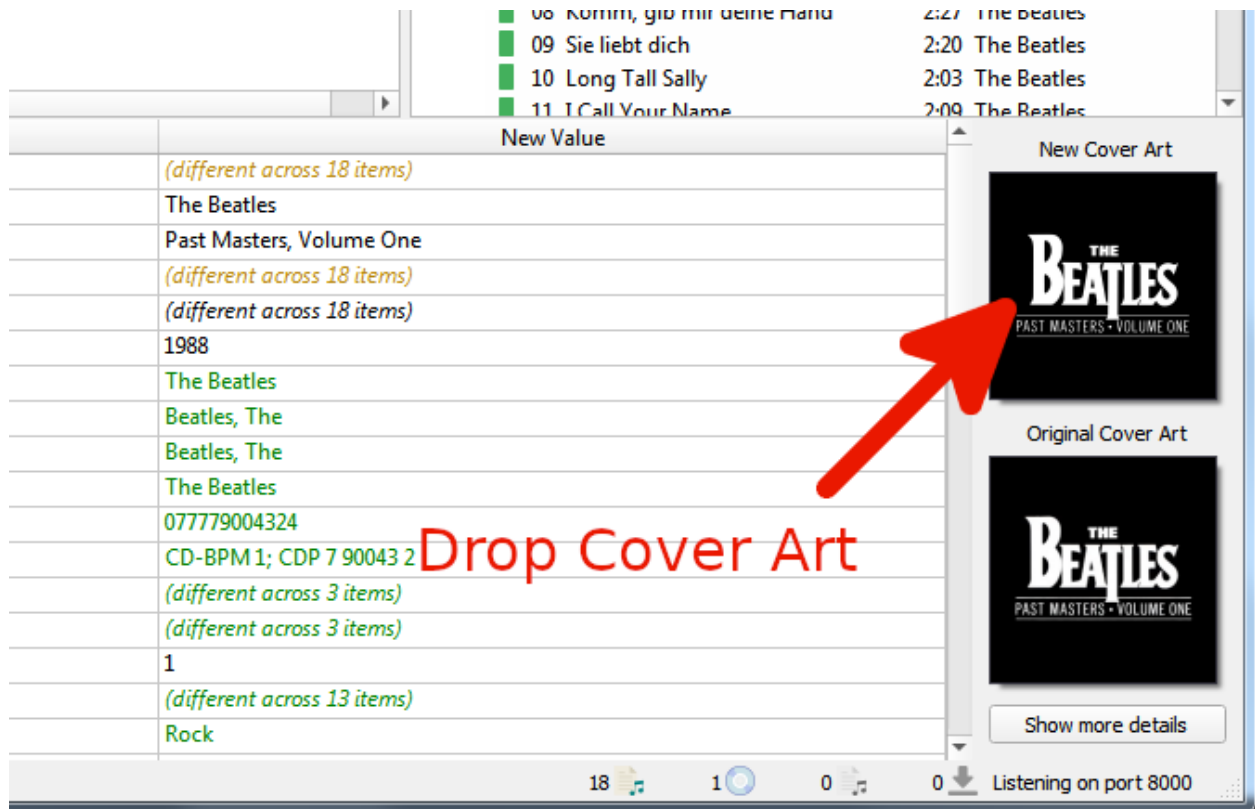
Right-clicking an item in the track list brings up a menu of commands, including “Info”, “Open in Player”, “Open Containing Folder”, “Search for similar tracks”, “Lookup in Browser”, “Generate AcoustID Fingerprints”, “Save” and “Remove”. In addition, you can re-run any associated plugins or scripts against 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.

10.3 Setting the Cover Art

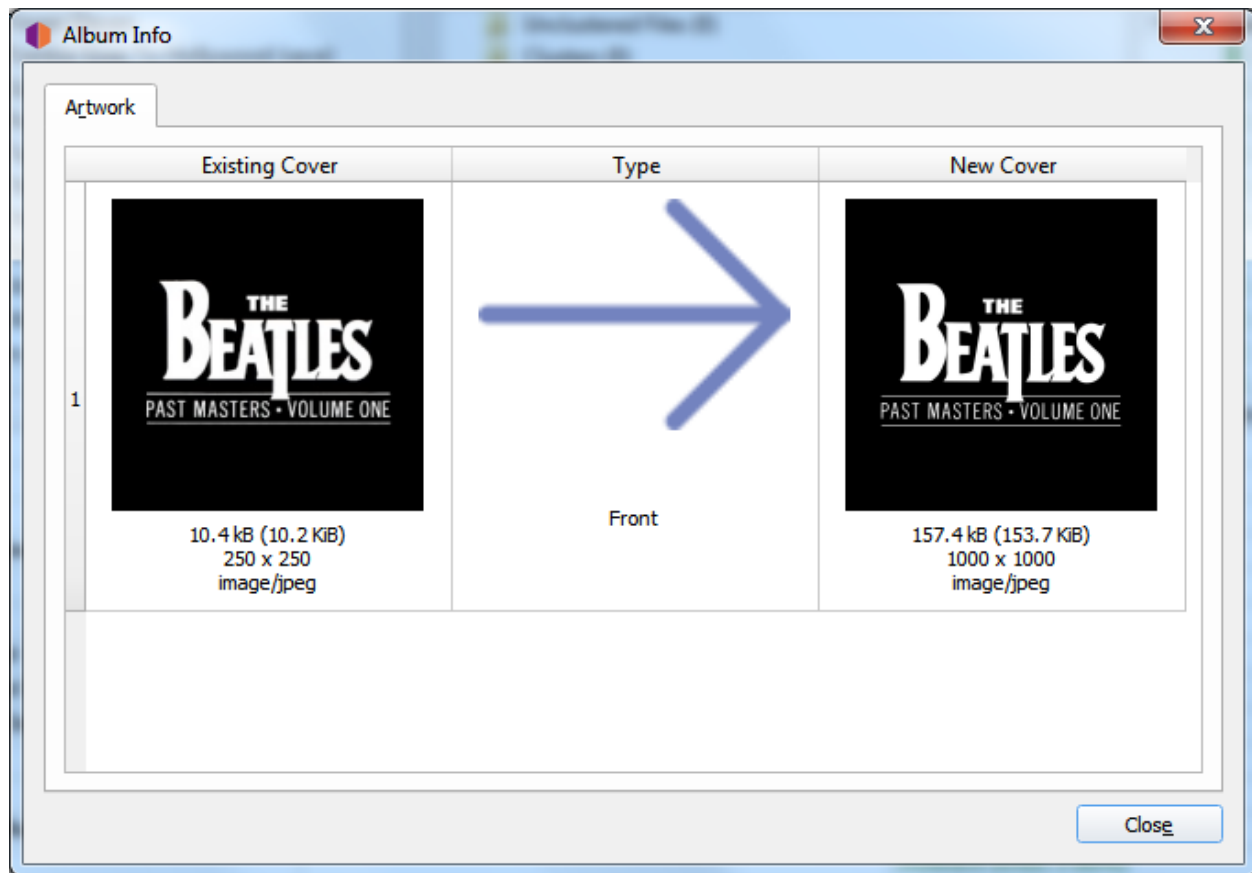
Depending on the option settings, you can change or confirm the cover art to save with a track or album.

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.



You can also choose a local file as cover art by right clicking on the image and selecting “Choose local file...” from the menu.

The menu also provides additional 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.

10.4 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
✓ 10 I - Me - We	2:03	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 the ripper log file is available*
3. *When files are grouped by album*
4. *When files are not grouped but have some metadata*
5. *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.

11.1 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

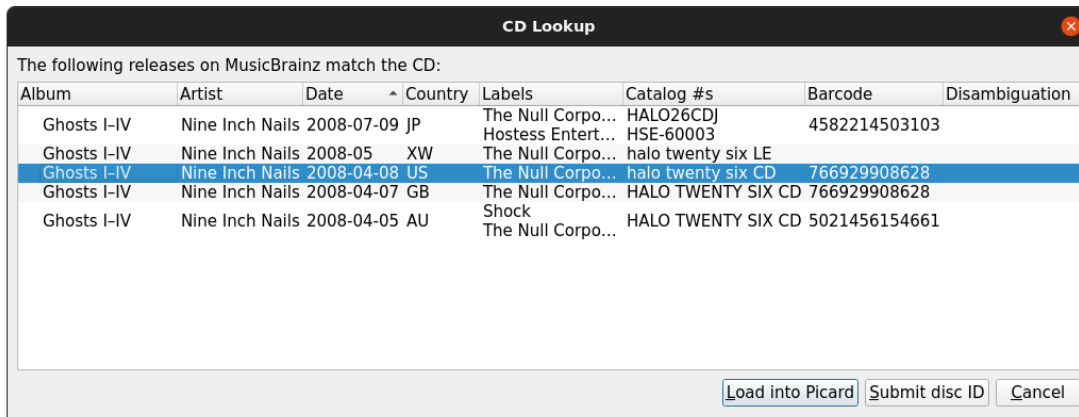
Extract the music filed from the CD using your favorite ripping program (e.g.: [Exact Audio Copy](#) for Windows, [X Lossless Decoder \(XLD\)](#) for macOS, or [Whipper](#) for Linux). The format for the output files depends on your personal preference and the formats supported by your player. A popular 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 or Ripper Log* section for detailed instructions.

3. Select the correct release

A list of all releases matching the toc of the CD will be displayed for selection, with an option to submit the disc id if none of the releases are a match to your CD. Before proceeding, please check to ensure that the release you select 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 acoustic features to AcousticBrainz or AcoustID fingerprints.



4. Load the files

Drag the files or folder from the browser to the “Unclustered Files” section 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.

11.2 When the ripper log file is available

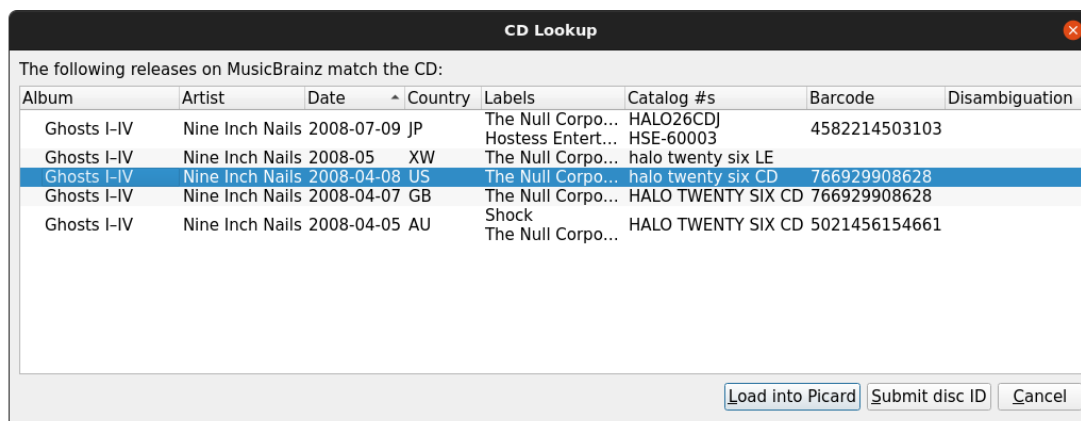
This option was added to Picard in version 2.8, and supports the use of log files produced by *supported popular CD file rippers*. Because the log files of these rippers contain sufficient information to generate the CD table of contents they can be used in place of reading the CD. As with reading the CD itself, this method 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. Lookup the CD on MusicBrainz

Use the ripper log file to look up the release automatically by selecting the “*Tools → Lookup CD → From CD ripper log file...*” command. This will open a file browser dialog to allow you to select the log file to process. See the *Lookup CD or Ripper Log* section for detailed instructions.

2. Select the correct release

A list of all releases matching the toc of the CD will be displayed for selection, with an option to submit the disc id if none of the releases are a match to your CD. Before proceeding, please check to ensure that the release you select 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 acoustic features to AcousticBrainz or AcoustID fingerprints.



3. Load the files

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

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.

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.

11.3 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” section 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.

11.4 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 processed.

1. Load the files

Drag the batch of files to process from the browser to the “Unclustered Files” section 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.

11.5 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 processed.

1. Load the files

Drag the batch of files to process from the browser to the “Unclustered Files” section 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.

OTHER PICARD TASKS

12.1 Attaching a Disc ID to a Release

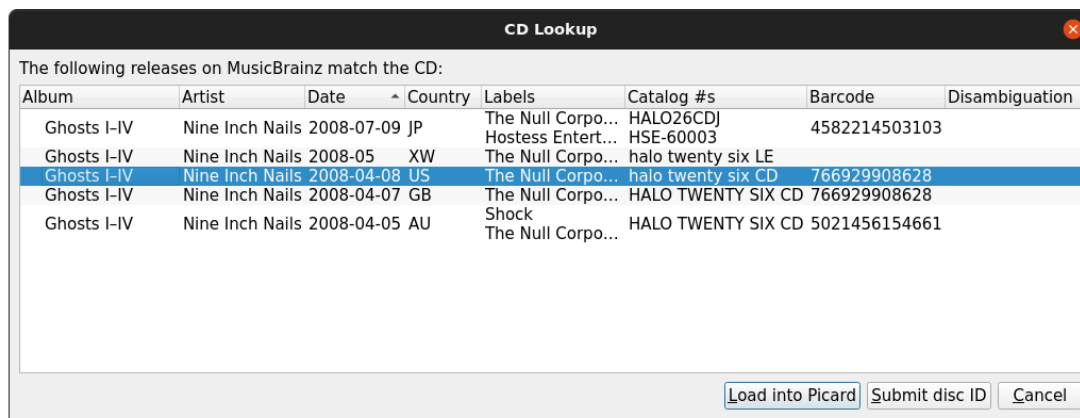
Disc IDs are very useful for identifying CDs and allowing MusicBrainz to know the length of tracks on a CD. Thus, it is very valuable to add them when submitting a new CD release or when you have a CD release that does not have a Disc ID attached.

Warning: Please do not add DiscIDs from CDs that are burned at home.

The steps to follow to submit a disc id are:

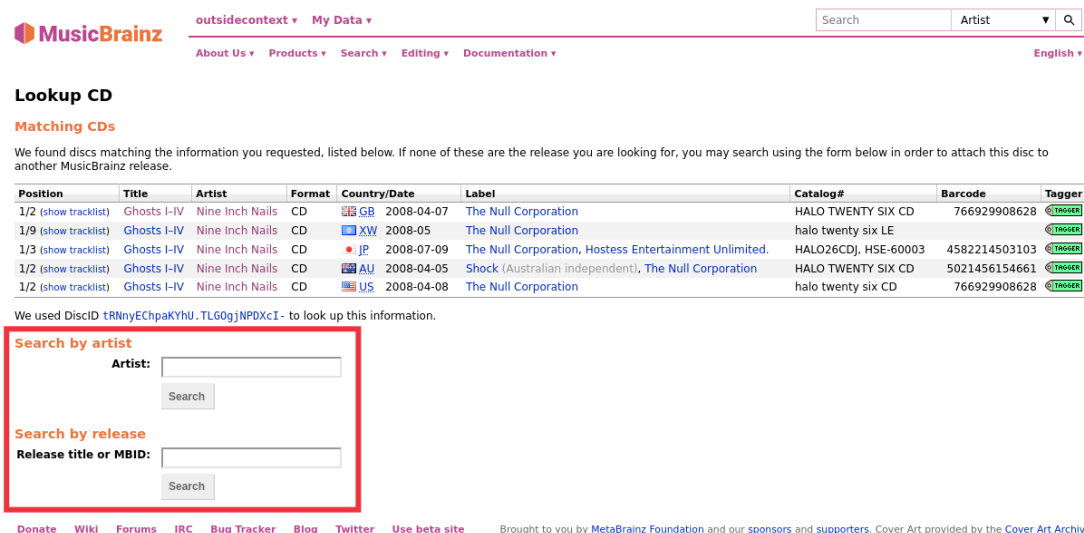
1. Lookup the CD

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. Alternately, you can use an EAC, XLD or Whopper ripper log file to lookup the CD using the “*Tools → Lookup CD → From EAC / XLD / Whopper log file...*” command. This will open a file browser dialog to allow you to select the log file to process. Either method will query the MusicBrainz database and display a list of matching releases.



2. Review list of matching releases

If the target release appears in this list, the disc id has already been attached and you do not need to do anything further. If there are no matches found or the desired target release does not appear in the list, use the “*Submit disc ID*” 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.



MusicBrainz outsidecontext ▾ My Data ▾ Search Artist ▾ Q

About Us ▾ Products ▾ Search ▾ Editing ▾ Documentation ▾ English ▾

Lookup CD

Matching CDs

We found discs matching the information you requested, listed below. If none of these are the release you are looking for, you may search using the form below in order to attach this disc to another MusicBrainz release.

Position	Title	Artist	Format	Country/Date	Label	Catalog#	Barcode	Tagger
1/2 (show tracklist)	Ghosts I-IV	Nine Inch Nails	CD	GB 2008-04-07	The Null Corporation	HALO TWENTY SIX CD	766929908628	Tagger
1/9 (show tracklist)	Ghosts I-IV	Nine Inch Nails	CD	XW 2008-05	The Null Corporation	halo twenty six LE		Tagger
1/3 (show tracklist)	Ghosts I-IV	Nine Inch Nails	CD	JP 2008-07-09	The Null Corporation, Hostess Entertainment Unlimited.	HALO26CDJ, HSE-60003	4582214503103	Tagger
1/2 (show tracklist)	Ghosts I-IV	Nine Inch Nails	CD	AU 2008-04-05	Shock (Australian Independent), The Null Corporation	HALO TWENTY SIX CD	5021456154661	Tagger
1/2 (show tracklist)	Ghosts I-IV	Nine Inch Nails	CD	US 2008-04-08	The Null Corporation	halo twenty six CD	766929908628	Tagger

We used DiscID tRNnyEChpaKYhU.TLG0gjNPDXcI- to look up this information.

Search by artist

Artist:

Search by release

Release title or MBID:

[Donate](#) [Wiki](#) [Forums](#) [IRC](#) [Bug Tracker](#) [Blog](#) [Twitter](#) [Use beta site](#) Brought to you by [MetaBrainz Foundation](#) and our [sponsors](#) and [supporters](#). Cover Art provided by the [Cover Art Archive](#).

Note: If you search by artist, use the radio button next to the artist’s name to select the desired artist. If you click on the artist’s name link, you will not be allowed to attach the disc id to any of the releases displayed.

3. Select the release

Select the desired target release from the list displayed by clicking the radio button next to the release, and then click the “Attach CD TOC” button below the list of releases. This will prepare an edit to attach the disc id to the release. You then need to add an appropriate edit note, and submit the edit.

Release Group: Splendid Genesis						
Splendid Genesis		Abstract Reason	XW	2015-01-24		[none]
Digital Media 1 (show tracklist)						
Release Group: Genesis						
Genesis		Genesis	GB	1983	Charisma	GENMC 1 [none]
Cassette 1 (show tracklist)						
Genesis		Genesis	DE	1983	Vertigo (British rock label)	814 287-2 042281428722
CD 1 (show tracklist)						
Genesis		Genesis	US	1983-10-03	Atlantic (Warner Music imprint)	
Digital Media 1 (show tracklist)						
Genesis		Genesis	XE	-	Charisma, Virgin (worldwide imprint of Virgin Records Ltd. and all its subsidiaries)	7 86436 2, GEN CD1 0077778643623
CD 1 (show tracklist)						
Genesis		Genesis	JP	1985-06-01	Vertigo (British rock label)	32PD-17 4988011303999
CD 1 (show tracklist)						
Genesis		Genesis	NL	1983-10-03	Mercury Records (or just "Mercury". A UMG imprint, do not use it for ©/® credits)	832 178-1 042281428715
12" Vinyl 1 (show tracklist)						
Genesis		Genesis	CA	1990	Atlantic (Warner Music imprint)	CD 80166 075678011627
CD 1 (show tracklist)						
Genesis (made in Japan)		Genesis	XE	-	Atlantic (Warner Music imprint)	7 80116-2, 814287-2 [none]
CD 1 (show tracklist)						

4. Add release if missing

If none of the releases displayed are correct, you have the option to add a new release (with some information automatically included). The disc id will automatically be attached to the new release when the edit is saved.

Release Group: Hesitation Marks

Hesitation Marks (24-bit deluxe edition)
US
2013-09-03
Columbia (imprint owned worldwide by S

Digital Media 1 (show tracklist)

Attach CD TOC

If you can't find what you're looking for, you can add a new release:

Add a new release

12.2 Submitting Acoustic Fingerprints

Acoustic fingerprints are very useful for identifying tracks and recordings, allowing them to be looked up in the MusicBrainz database. Thus, it is very valuable to add them when you are tagging files. Note that an acoustic fingerprint is **not** an AcoustID. Please see the [Understanding Acoustic Fingerprinting and AcoustIDs](#) tutorial for additional information.

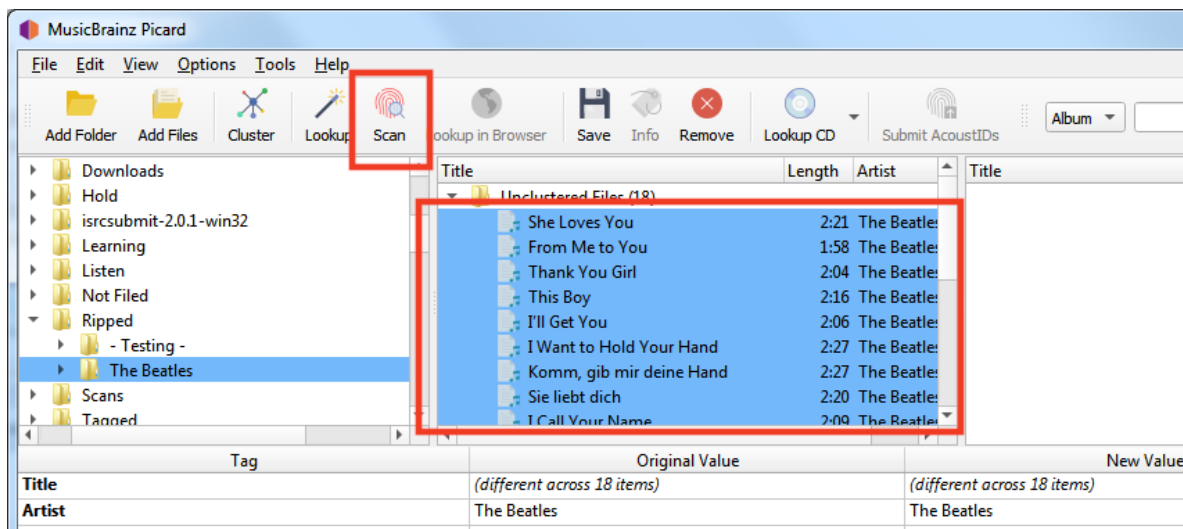
Note: When using Picard to submit acoustic fingerprints, it is recommended to enable the Fingerprint column in the table view in the right-hand pane. This is done by right-clicking the column header and checking the box beside “Fingerprint status”. This will display an icon indicating whether the AcoustID was calculated and whether it ready for submission (red = unsubmitted, grey = already submitted).

There are two methods for submitting acoustic fingerprints, depending on the workflow that you are using to identify the releases that you are tagging. Note that both methods require that you first match your audio files to release and track information from the MusicBrainz database. See the [Retrieving Album Information](#) and [Matching Files to Tracks](#) sections for more information about retrieving release information and matching audio files to releases.

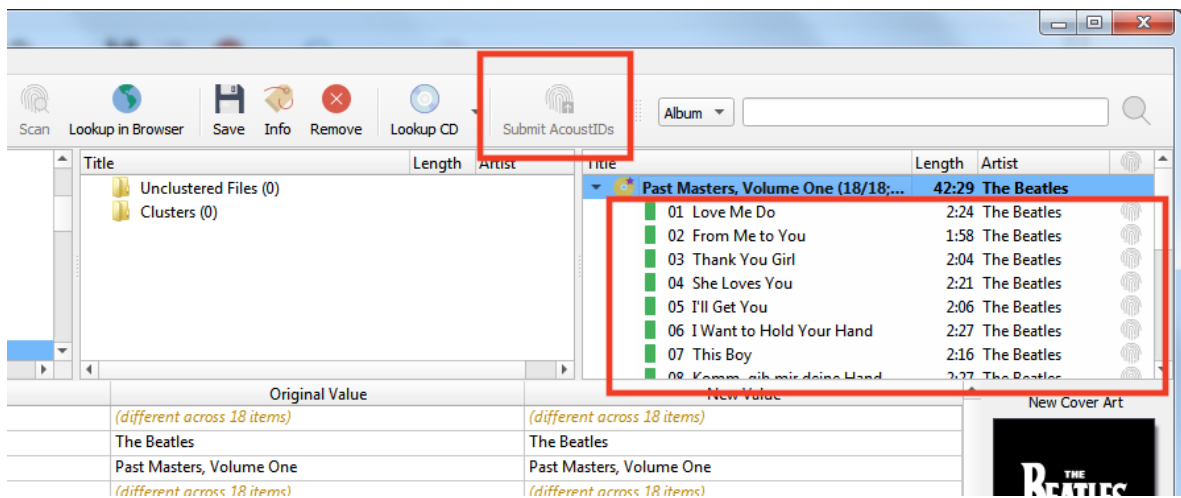
The steps to follow to submit acoustic fingerprints for each of the two workflows are:

12.2.1 Submitting when using Scan to identify the release

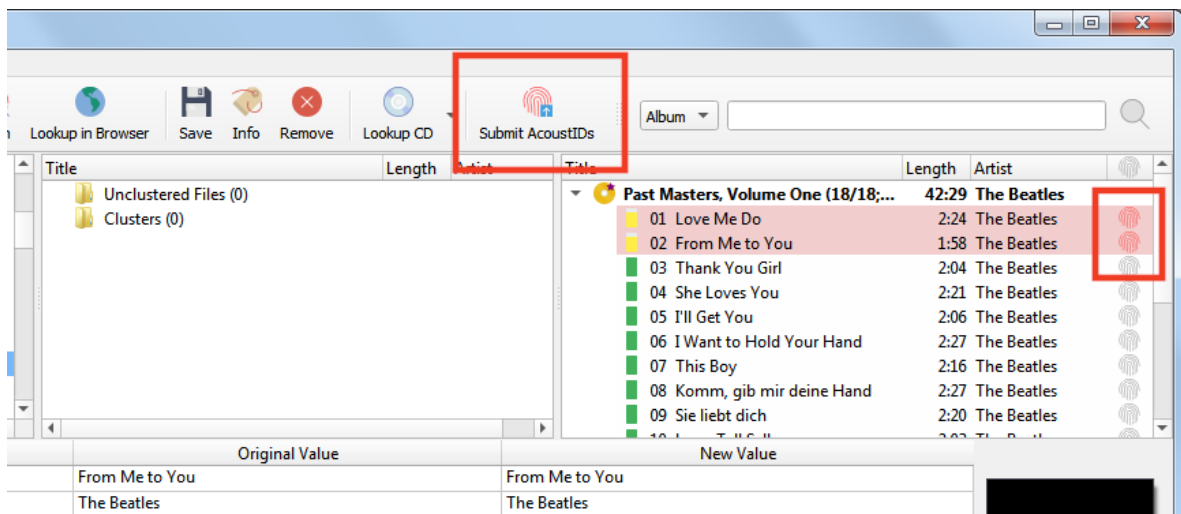
1. Load files into the clustering pane. Select the files and click the “Scan” button, or select “Tools → Scan”.



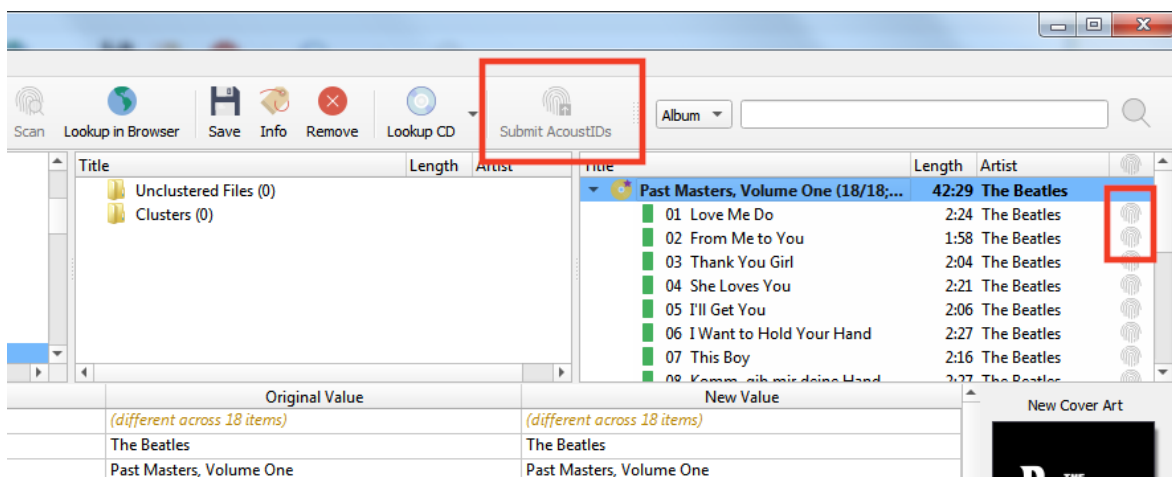
- If the files are matched to a track and move to the right-hand pane, they already exist in the AcoustID database and do not need to be re-submitted. The “Submit” button will remain disabled.



- If the files are not matched, or you manually move them to match to a different track they could be submitted. The AcoustID icon for the tracks will show up in red (i.e.: unsubmitted status) and the “Submit” button will be enabled.

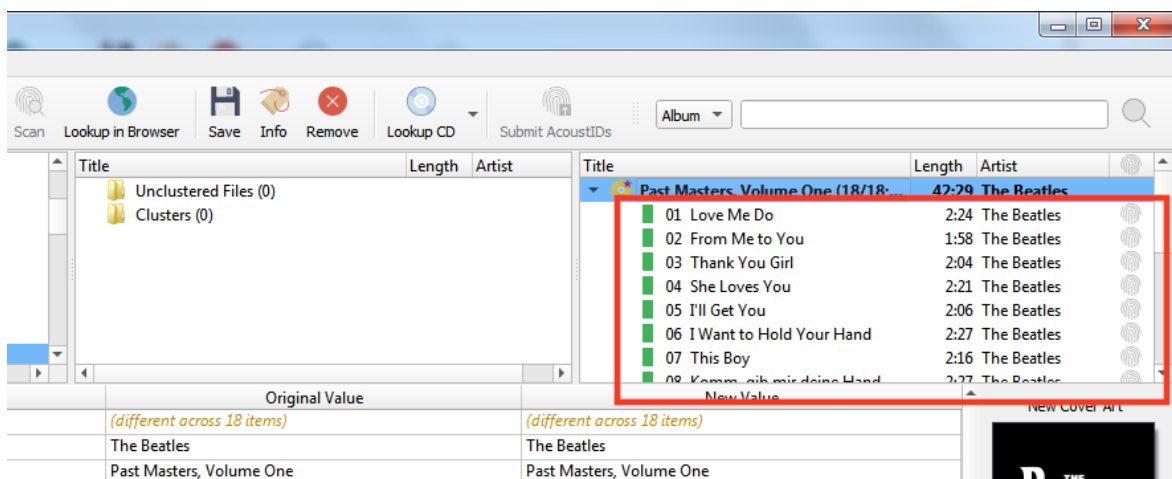


4. Clicking the “Submit” button will only submit the fingerprints for the files identified in Step 3. The AcoustID icon for the tracks will change to grey (i.e.: submitted status) and the “Submit” button will be disabled.

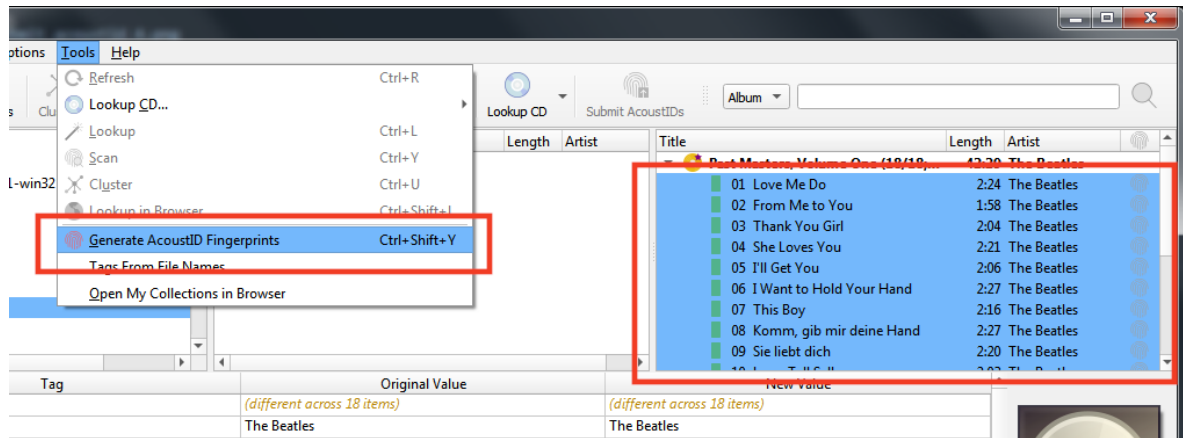


12.2.2 Submitting when not using Scan to identify the release

1. Make sure that the files are properly matched to tracks on a release in the right-hand pane.

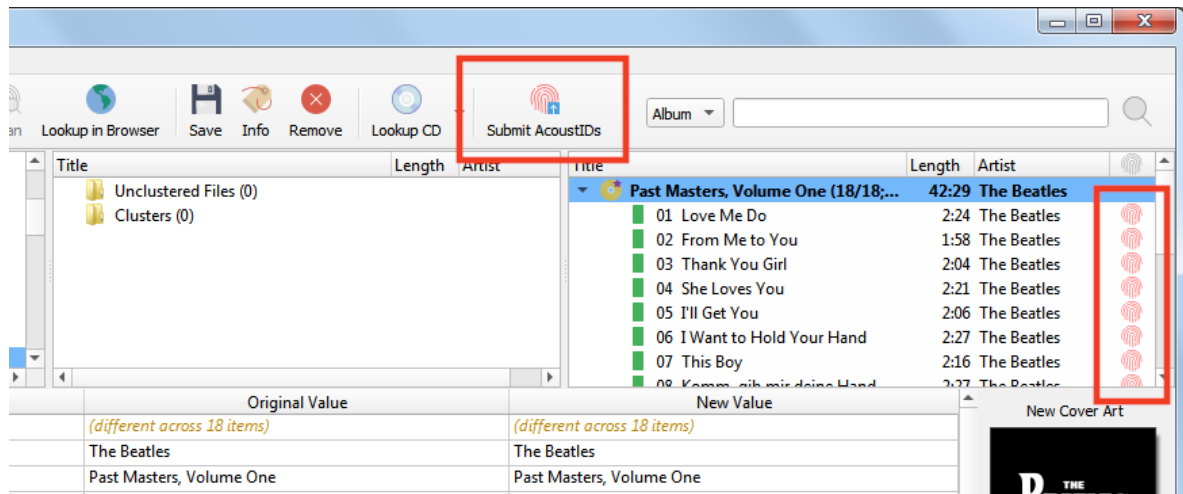


2. Select the files in the right-hand pane and select “Tools → Generate AcoustID fingerprints”. This will calculate the acoustic fingerprints for the selected files.

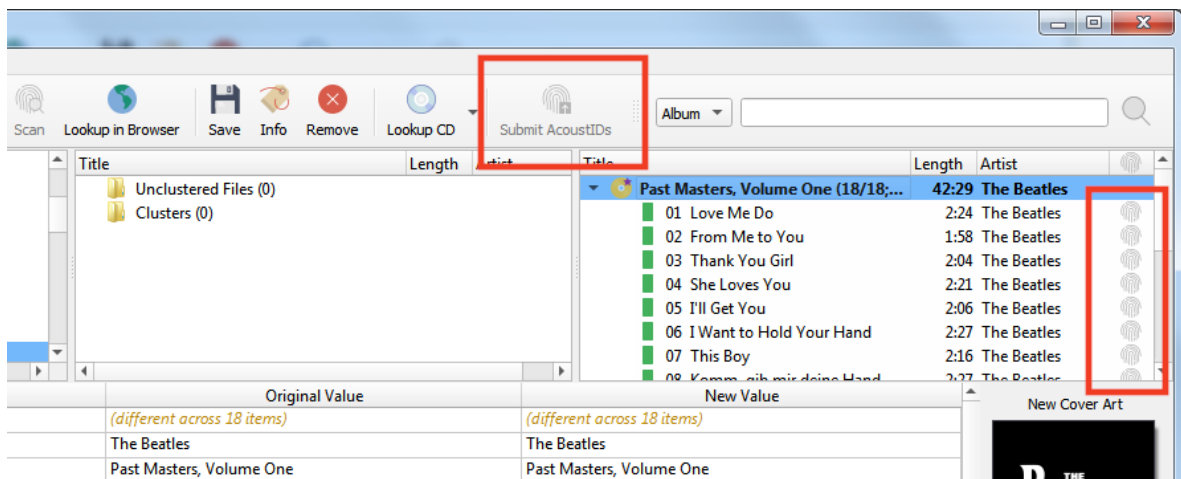


Note: The “Generate AcoustID fingerprints” action button can be added to the button bar by changing the settings in the User Interface options.

The AcoustID icon for the tracks will show up in red (i.e.: unsubmitted status) and the “Submit” button will be enabled.



3. Clicking the “Submit” button will submit the fingerprints for the files. The AcoustID icon for the tracks will change to grey (i.e.: submitted status) and the “Submit” button will be disabled.

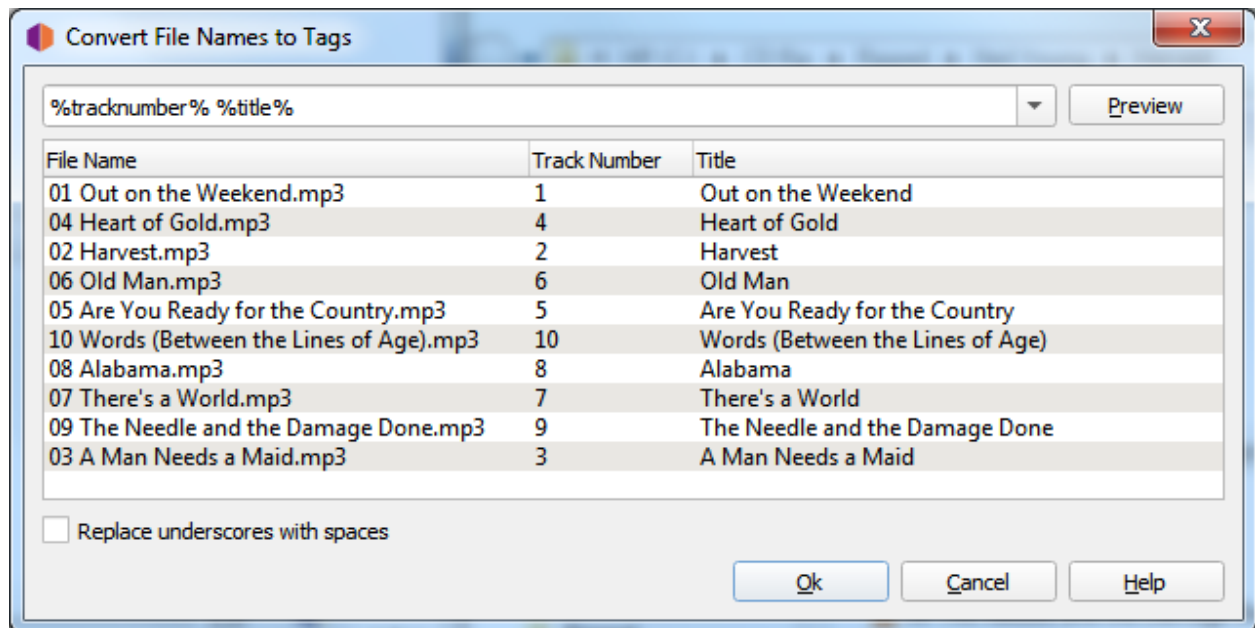


12.3 Generating tags from file names

Sometimes files have poor quality tags or no tags at all, but the file names are well structured and follow a pattern. In this case you can use “Tools → Tags From *File Names...*” to generate the tags from the file names.

12.3.1 Basic usage

To use this tool, select one or more files loaded into Picard and open the Tags From File Names dialog from the menu at “Tools → Tags From *File Names...*”. The dialog will show you a list of filenames and an input field at the top where you can enter a matching pattern.



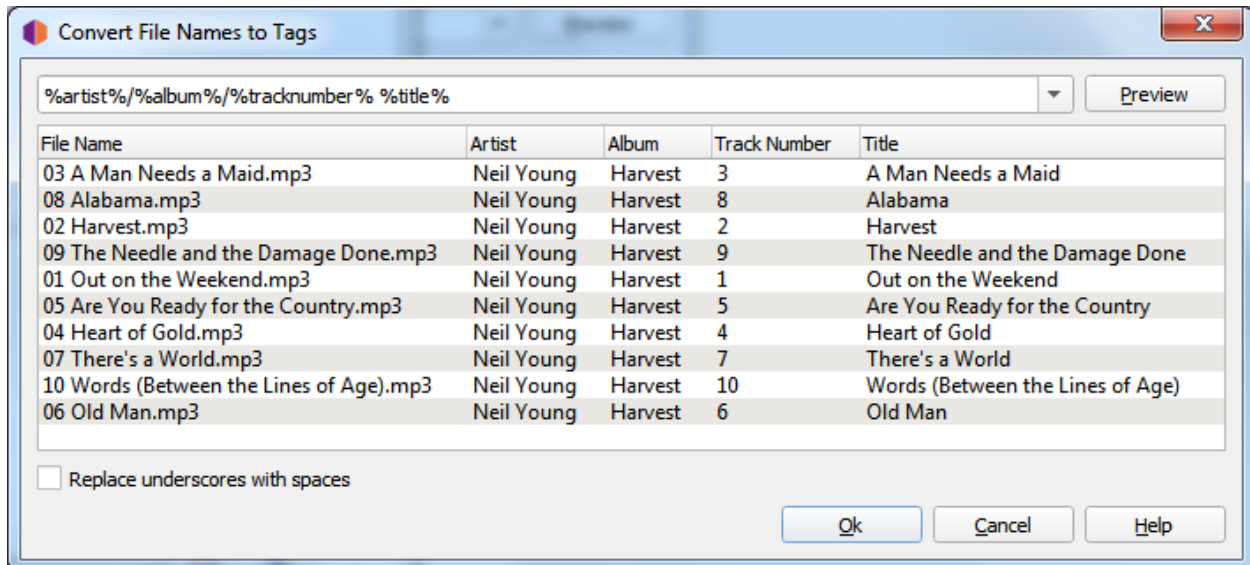
The matching pattern can consist of Picard tag names enclosed in % signs and other characters that are matched verbatim. For the tag names you can use predefined names such as %artist%, %album%, or %title% (see [Tags & Variables](#)) or use custom names. There are a few predefined patterns available to select from, but you can also adjust them or set your own.

If your files for example consist of a track number and track title separated by a space (e.g. 04 Heart of Gold.mp3) you can use the matching pattern %tracknumber% %title%. Should the track number and title be separated by for example a hyphen like 04 - Heart of Gold.mp3 the pattern needs to also include this separator, like %tracknumber% - %title%.

Clicking on the “Preview” button next to the matching pattern will show a preview of the extracted tags for each file name. Once you are satisfied with the result, you can accept the changes with the “Ok” button. The changed tags will be set for the files. Note that the changes will not be saved automatically, you still need to save the files if you want the tags to be written (see [Saving Updated Files](#)).

12.3.2 Matching folders

The pattern can also match the parent folders of the file. To match for folders use a slash (/) as separator. If for example the file is in a folder named after the album, which in turn is inside a folder named after the artist (i.e. Neil Young/Harvest/04 Heart of Gold.mp3) you could match the artist, album, track number and title with a pattern of %artist%/%album%/%tracknumber% - %title%.



12.3.3 Replace underscores with spaces

Sometimes files have been named without spaces and use underscores instead. For example a file could be named 04_Heart_of_Gold.mp3. By default the title would get extracted as “Heart_of_Gold”. In this case enable the checkbox “Replace underscores with spaces” and use a pattern like %tracknumber%_%title% to extract the title properly as “Heart of Gold”.

12.3.4 Ignoring parts of the file name

Sometimes you don’t want to include parts of the file name in your tags and just want to ignore them. The pattern must always match the entire file name, though. In this case you can use a hidden variable for the parts of the file name you do not want to match to an actual tag. Hidden variables start with an underscore like %_dummy%. This variable will still be available on the file for *Scripting*, but will not get written to the actual file tags on saving.

One example might be if you want to extract only the track number from a file name like 04 Are You Ready for the Country_.mp3. Maybe the track number tag is missing in the file, but the title tag is already properly set. You could use the pattern %tracknumber% %_filetitle%. This would extract the tracknumber tag properly, but would extract the rest of the file name to a hidden variable %_filetitle% which would

not get written to the file tags. The name `%_filetitle%` is arbitrarily chosen, it just needs to start with an underscore.

Note: Parsing hidden variables from file names is supported since Picard 2.5. Earlier versions would create an actual tag which would get stored to the tags. If you are using a Picard version older than 2.5 you will need to remove the unwanted tags before saving the files.

12.4 Submitting Cluster as a Release

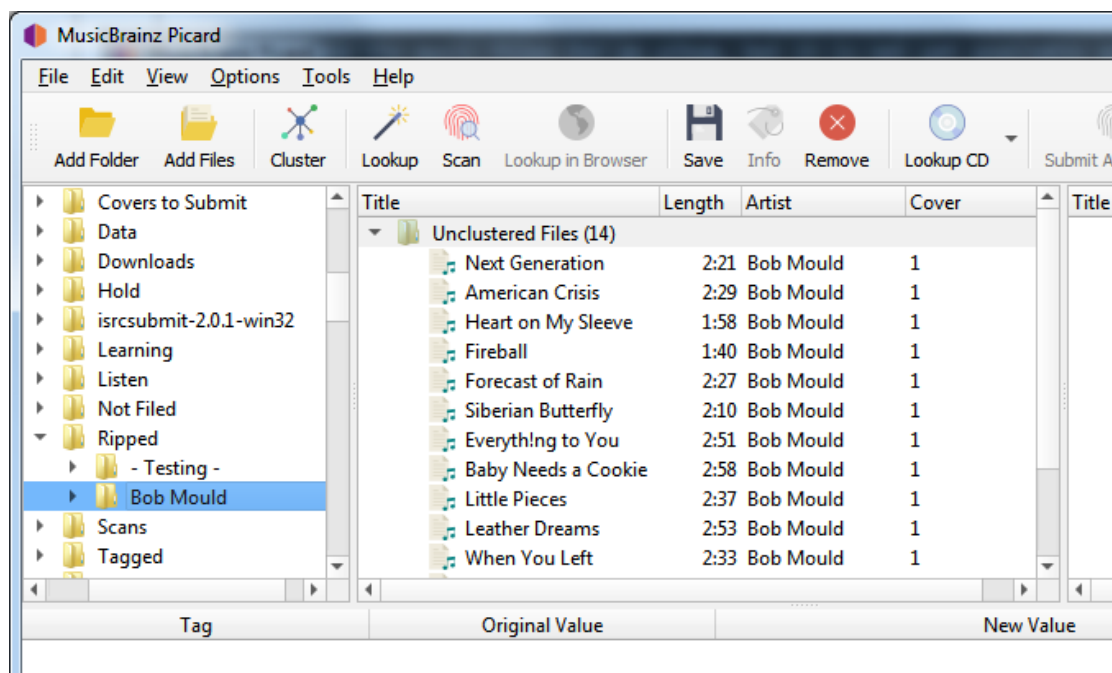
Picard can assist you in submitting information to the MusicBrainz database by automatically populating the submission form on the website with data from your files. This is typically used when you have the music files for an album, but it is not yet available on MusicBrainz.

12.4.1 Submitting multiple tracks as a cluster

To use this functionality, the steps to follow are:

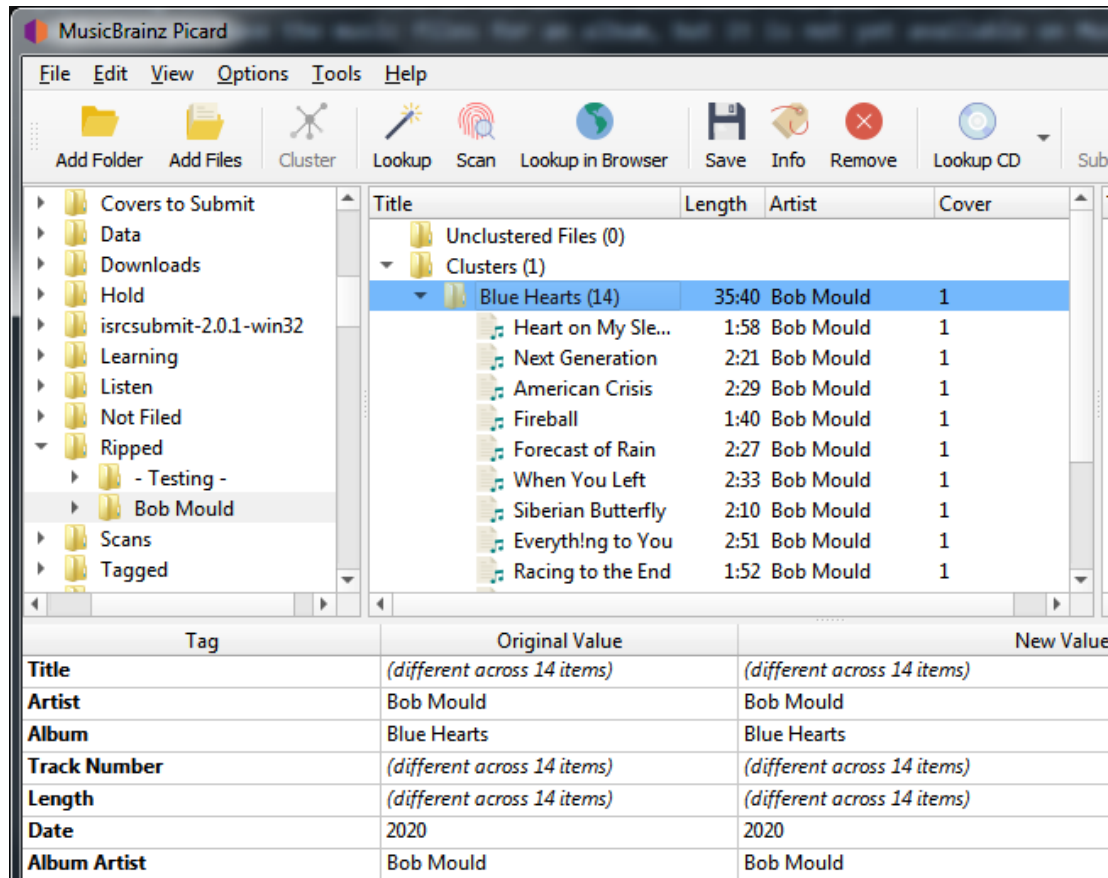
1. Load the files

Drag the batch of files to process from the browser pane to the “Unclustered Files” section in the clustering pane.



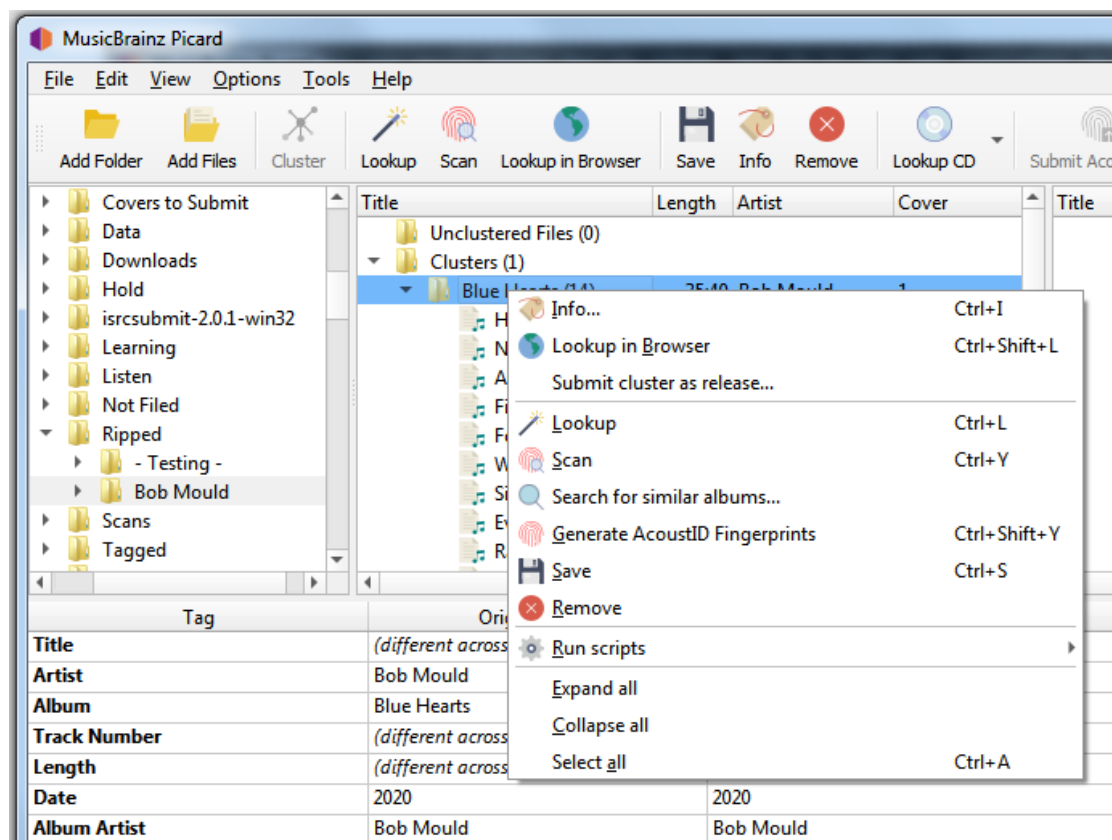
2. Cluster the files

Select the files in the clustering pane and combine them into album clusters using the “Tools → Cluster” command. Picard will attempt to cluster the files based on their existing metadata. Depending on the quality of the metadata, you may need to manually add items to the cluster or remove items from the cluster to ensure that it is complete for the album, and does not contain any additional files.



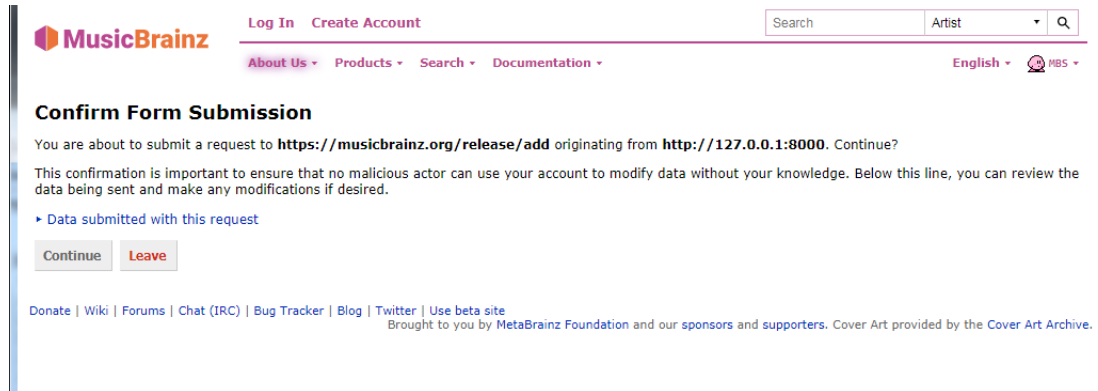
3. Submit the cluster

Once you have the proper files in the cluster and it is complete for the album, you can submit it to MusicBrainz by selecting the cluster and right-click to bring up the context menu. From there you should see an option to “*Submit cluster as release...*”.



4. Confirm submitted information

When the option is selected, the system will submit a request to add the information to MusicBrainz, and you will be presented with a confirmation screen in your browser. You can see the information that will be submitted by expanding the “Data submitted with this request” link.



5. Complete the submission

Selecting *Continue* will open an “Add Release” edit screen with the fields populated with your information. From here you can check and submit your edit as if you had entered all of the information manually.

Note: Before submitting the edit, you should check that all of the information has been correctly entered in accordance with the [MusicBrainz Style Guides](#) and that the artist and release groups have been matched to existing items as appropriate. You should also add an edit note citing the source of the information.

12.4.2 Submitting a single track

You can also use this feature to submit a single track as a release or as a standalone recording. To do this, right-click the file and select either “*Submit file as standalone recording...*” or “*Submit file as release...*” as appropriate. Again, be sure to confirm the information has been correctly entered in accordance with the [MusicBrainz Style Guides](#) and that the artist and release groups have been matched to existing items as appropriate. You should also add an edit note citing the source of the information.

OPTION PROFILES

As of version 2.7, Picard supports multiple profiles that can allow the user to quickly switch between option settings.

13.1 How Option Profiles Work

A profile is defined by a set of options it manages. For example, one profile may include settings for file naming such as the target directory and which file naming script to use, while another profile may include different settings for the same options or different options entirely (or some of each). Profiles are stacked and processed in the order specified by the user, from top to bottom with the lowest level being the system's "user settings" profile. Each user-defined profile can be enabled or disabled independently from the other user-defined profiles. The system's "user settings" profile is always enabled and includes all options.

When an option value is retrieved as part of Picard's processing, it comes from the first enabled profile in the stack that manages that option. Initially, the profile stack contains only the system's "user settings" profile, which holds the default settings for the user.

13.2 Example of Using Profiles

For this example, the user would like to define a set of options with alternate values, in this case a target directory where audio files are saved (option `move_files_to`).

The user creates a new profile (named "TargetMyDir"), adds the option `move_files_to` to it, and enables this profile. The stack is now:

<pre>[x] TargetMyDir move_files_to [x] user settings move_files_to [plus all other settings]</pre>

They change the value of `move_files_to` (to "target_my_dir") for this new profile.

Since the profile "TargetMyDir" is enabled, the value for `move_files_to` is retrieved from this profile. The "user settings" still has the old `move_files_to` value.

Now the user wants to work on another set of music files, wanting to disable `windows_compatibility` for this set and save them to the “not_for_windows” directory.

They create a new profile (named “ByeByeWin”), add options `move_files_to` and `windows_compatibility`, and enable it. Now the stack looks like:

```
[x] ByeByeWin      move_files_to  windows_compatibility
[x] TargetMyDir    move_files_to
[x] user settings  move_files_to  windows_compatibility  [plus all
↳other settings]
```

They change the values of `move_files_to` (to “not_for_windows”) and `windows_compatibility` (to false) for this new profile. Now when they process their files, the files are saved to the “ByeByeWin” `move_files_to` directory, with `windows_compatibility` = false.

Now the user wants to save files to the “TargetMyDir” target directory again, with their usual options. To do this they simply disable the “ByeByeWin” profile (which can later be re-enabled if needed). The stack looks like:

```
[ ] ByeByeWin      move_files_to  windows_compatibility
[x] TargetMyDir    move_files_to
[x] user settings  move_files_to  windows_compatibility  [plus all
↳other settings]
```

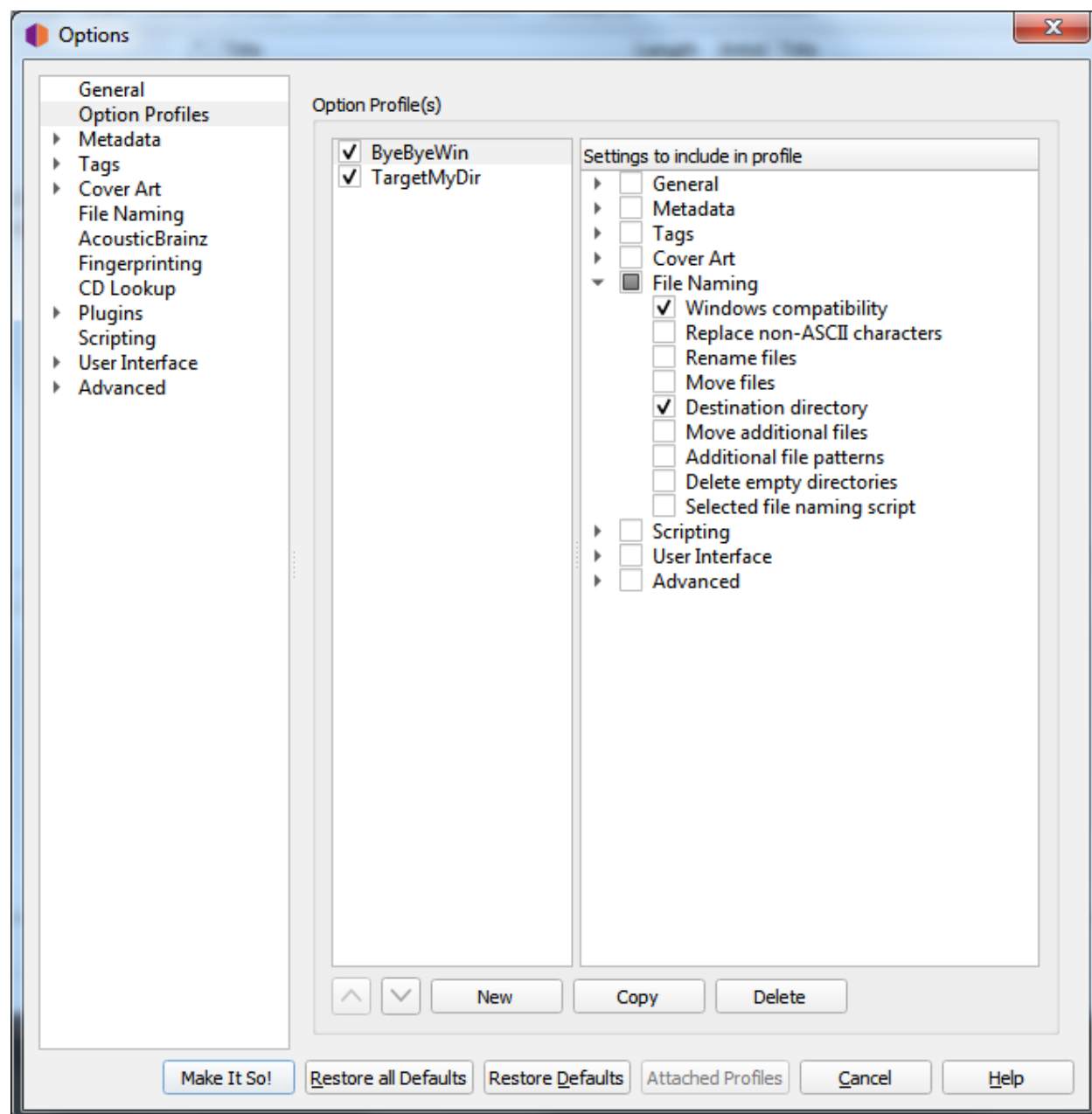
Finally, to return to their usual output directory the user only has to disable the “TargetMyDir” profile so the stack is:

```
[ ] ByeByeWin      move_files_to  windows_compatibility
[ ] TargetMyDir    move_files_to
[x] user settings  move_files_to  windows_compatibility  [plus all
↳other settings]
```

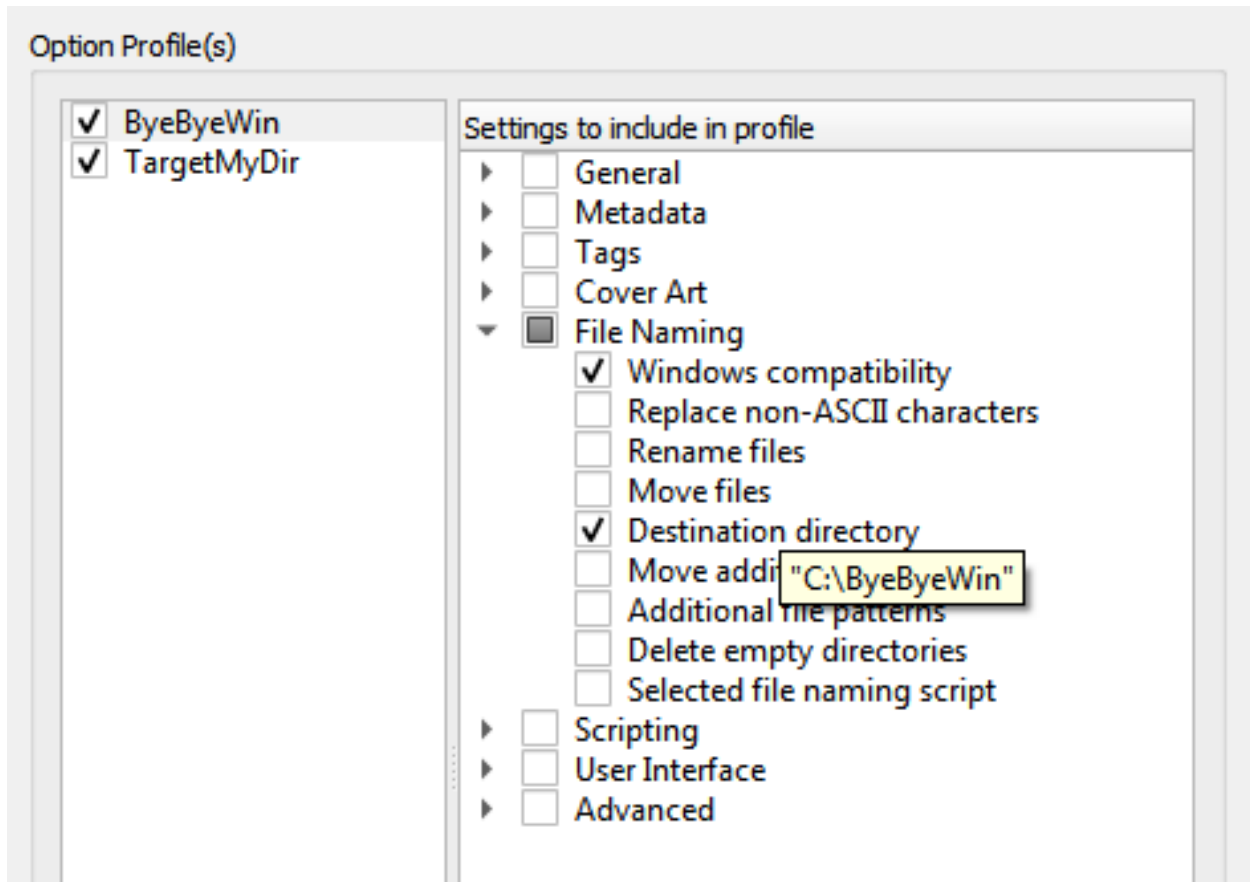
13.3 Managing Option Profiles

All option profile management is done within the Option Profiles page available from the “*Options* → *Options...*” item on the menu bar. From this screen you will be able to add, copy, edit, remove, enable and disable profiles, as well as setting the order of the profile stack.

Initially, the list of profiles will be empty. To create a new profile click on the *New* button. This will create a profile with no options selected for the profile to manage. To rename the profile, right-click on the profile name and select the “*Rename profile*” command. The list of options that the profile is to manage are selected from the list in the right-hand pane. Options can be selected either by group or individually. The groups can be expanded to see the individual options belonging to that group.



You can see the value currently assigned to a profile's option setting by hovering your cursor over the setting in the list. The value will be displayed as a tooltip for the setting.



The profile stack order can be rearranged either by selecting a profile and using the up and down arrow buttons below the list, or by dragging the profile to a new position in the stack. Profiles are enabled when the box beside the profile's name is checked.

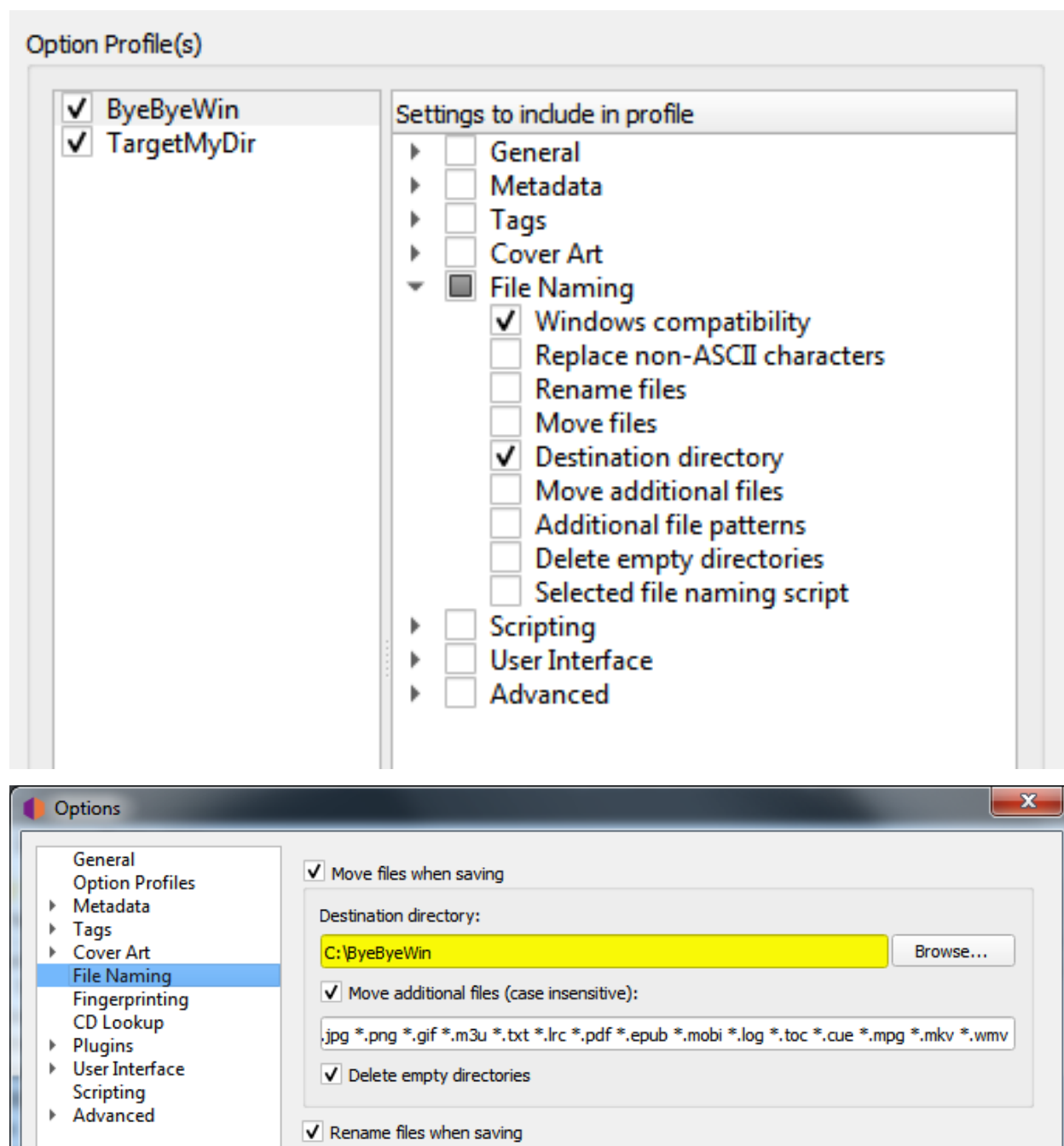
Changes made to a profile's options settings, enabled status, or position in the profile stack will be reflected in the option settings displayed on the other pages. Options that are controlled by an enabled profile will be shown as highlighted. Hovering your cursor over the highlighted option will identify which profile currently controls the setting. Settings are always displayed based on the first enabled profile in the profile stack, which corresponds to the setting that will be used during processing.

You can also quickly enable or disable a profile (but not change the order of the profile stack), using the *"Options → Enable/disable profiles"* item in the menu bar on Picard's main screen.

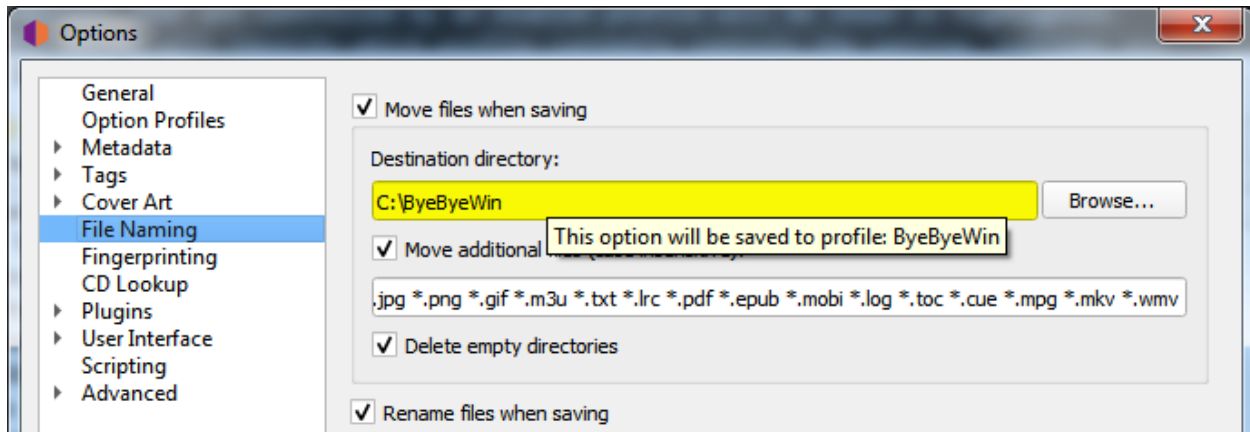
When you click the *Make It So!* button, in addition to saving your updated profile configuration all highlighted options will be saved to the first enabled profile in the profile stack that controls that option. All other options will be saved to the "user settings" as before. This is described in more detail in the following section.

13.4 Saving Profile Option Settings

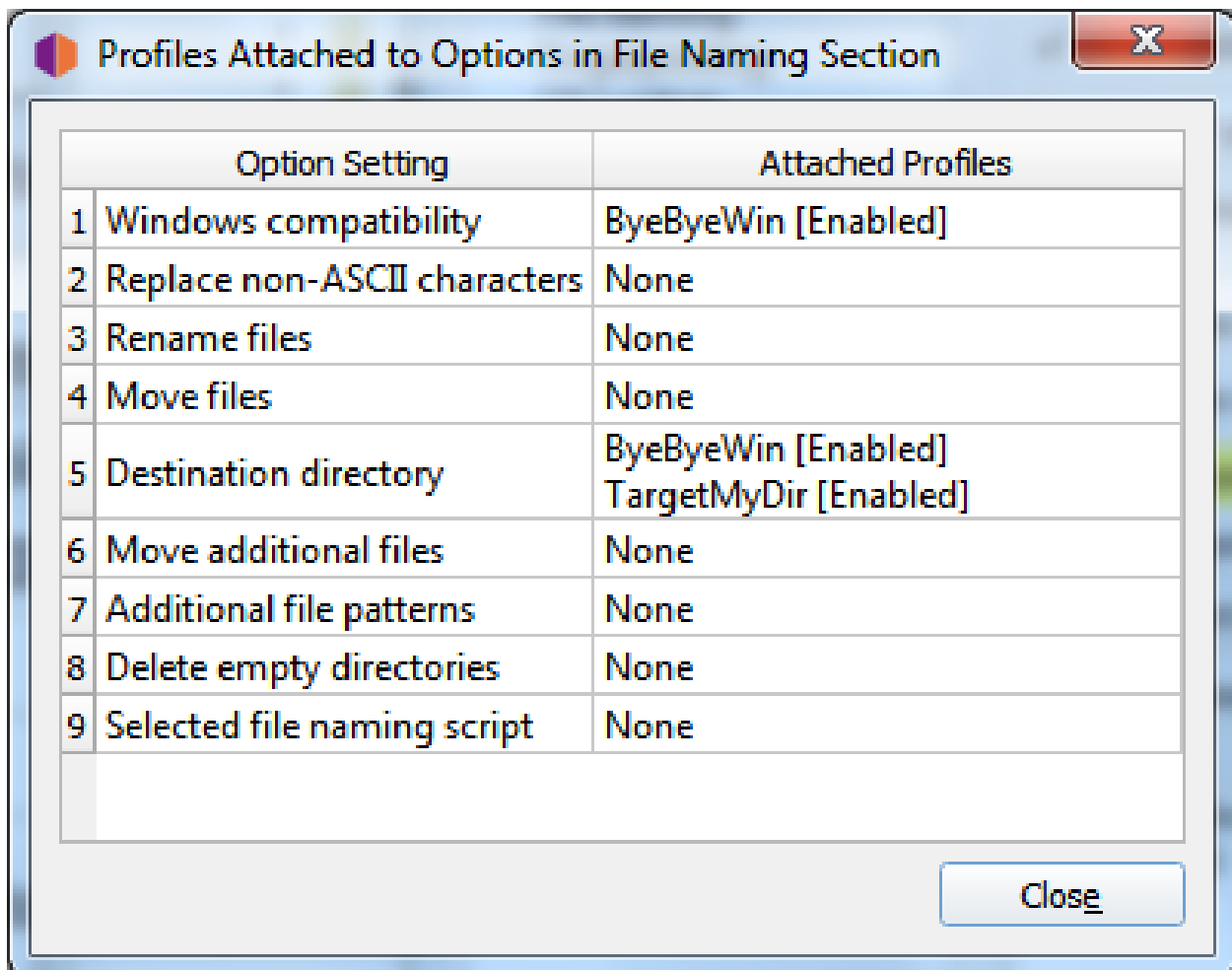
To save a value to a profile option setting, simply ensure that the the target profile is the first enabled profile in the profile stack, make the desired changes (the options should be highlighted), and then click the *Make It So!* button.



Remember, all highlighted options will be saved to the first enabled profile in the profile stack that controls that option. All other options will be saved to the “user settings” profile which is the user’s normal settings, and includes all options. You can confirm which profile a highlighted option will update by hovering your cursor over the option.



From the pages which contain options that can be included on a profile, you will also be able to see which profiles, if any, manage any of the options on the page. This is done by clicking the *Attached Profiles* button.



This lists the attached profiles in the order in which they appear in the profile stack, and whether or not the profile is enabled. If the page does not contain any options that could be managed by a profile, the *Attached Profiles* button will be disabled.

Warning: It is important to understand that when you click the *Make It So!* button **all** of the option settings on **all** pages will be saved. If an option is managed by one or more profiles that are currently enabled, the option will be highlighted and it will be saved to the **first** enabled profile in the profile stack that manages the option. If there are no enabled profiles that manage the option, the option will not be highlighted and it will be saved to the “user settings” profile which is the user’s normal settings, contains all options, is at the bottom of the profile stack, and is always enabled. The “user settings” profile cannot be modified is not shown in the profile management page.

COMMAND AND BATCH PROCESSING

As of version 2.9, Picard will try to only run a single instance of the program at a time. When the program is started, it checks to see if there is another instance of that version, configuration file and plugin startup status -P already running. If the same version is already running, any files or directories specified on the command line of the new instance, along with any executable commands specified with the -e or -exec options will be passed to the already running instance for processing and the new duplicate instance will be shut down. This allows batch processing of files to be initiated automatically from other processes. If there is no instance of that version already running, Picard will start normally.

For example if there is an instance of Picard running and a second instance is started with the command line:

```
picard -e load mbid://release/dbd0ce67-cae6-33eb-8f5a-1143a30c2353
```

the load command will be passed to the running instance to load the specified release, and the second instance will be closed.

This allows the user to set up dynamic batch processing of commands to automate the tagging process, especially when used with the FROM_FILE command to load a standard processing command sequence such as:

```
CLUSTER  
LOOKUP_CLUSTERED  
SAVE_MATCHED  
REMOVE_SAVED  
REMOVE_EMPTY
```

or:

```
LOOKUP_CD path/to/ripper.log  
SAVE_MATCHED  
FINGERPRINT  
SUBMIT_FINGERPRINTS  
REMOVE_SAVED  
REMOVE_EMPTY
```

or even something like:

```
# Load a directory of files to process
LOAD path/to/directory/of/unprocessed/files

# Try clustering and lookup the clusters first
CLUSTER
LOOKUP_CLUSTERED

# Save matched clusters
SAVE_MATCHED

# Calculate and submit fingerprints for the matched files
FINGERPRINT
SUBMIT_FINGERPRINTS

# Clean up and remove the saved files
REMOVE_SAVED
REMOVE_EMPTY

# Try scanning the remaining files to find matches
SCAN

# Save matched files from the scans
SAVE_MATCHED

# Clean up and remove the saved files
REMOVE_SAVED
REMOVE_EMPTY

# Any files remaining in the cluster pane could not be
# matched automatically
```

Please see the *Executable Commands* section for details regarding the commands available for execution.

14.1 Executable Commands

Picard can accept commands for processing by specifying them on the command line using the `-e` option or loading them from a text file. Commands are case-insensitive, and are processed sequentially in the order that they are received. The executable commands that Picard recognizes are:

14.1.1 CLEAR_LOGS

Usage: **CLEAR_LOGS**

Implemented: Picard 2.9

Clear all entries from Picard's log. This is the equivalent of clicking the *Clear Log* button from the log viewing screen opened using the *"Help → View Error/Debug Log"* command.

14.1.2 CLUSTER

Usage: **CLUSTER**

Implemented: Picard 2.9

Cluster all files in the cluster pane. This is the equivalent of using the *"Tools → Cluster"* command.

14.1.3 FINGERPRINT

Usage: **FINGERPRINT**

Implemented: Picard 2.9

Calculate acoustic fingerprints for all (matched) files in the album pane. This is the equivalent of using the *"Tools → Generate AcoustID Fingerprints"* command.

14.1.4 FROM_FILE

Usage: **FROM_FILE <file path>**

Implemented: Picard 2.9

Load commands from a file. The file path can be either an absolute or relative path to a text file containing the commands to be executed. Each command to be processed must be on a separate line along with its arguments (if applicable). Blank lines and

lines beginning with an octothorpe (#) are ignored. Command files can include other command files by specifying them with another FROM_FILE command. Circular references (by including a command file that is currently being processed) are ignored and will be logged as a warning.

For example, you may have a file named `commands.txt` containing the standard commands that you want to use when processing each directory, such as:

```
# Try clustering and lookup the clusters first
CLUSTER
LOOKUP_CLUSTERED

# Save matched clusters
SAVE_MATCHED

# Calculate and submit fingerprints for the matched files
FINGERPRINT
SUBMIT_FINGERPRINTS

# Clean up and remove the saved files
REMOVE_SAVED
REMOVE_EMPTY

# Try scanning the remaining files to find matches
SCAN

# Save matched files from the scans
SAVE_MATCHED

# Clean up and remove the saved files
REMOVE_SAVED
REMOVE_EMPTY

# Any files remaining in the cluster pane could not be
# matched automatically
```

You could then process a directory by starting Picard with the command:

```
picard -e LOAD path/to/directory/of/unprocessed/files -e FROM_FILE_
↳ commands.txt
```

14.1.5 LOAD

Usage: **LOAD** <supported MBID/URL or path to a file/directory>

Implemented: Picard 2.9

Load one or more files/directories/MBIDs/URLs to Picard. This is similar to including the file, directory path, URL or MBID on the command line.

Files and directories are specified including the path (either absolute or relative) to the file or directory, and may include drive specifiers. They can also be specified using the `file://` prefix. URLs are specified by using either the `http://` or `https://` prefix. MBIDs are specified in the format `mbid://<entity_type>/<mbid>` where `<entity_type>` is one of “release”, “artist” or “track” and `<mbid>` is the MusicBrainz Identifier of the entity.

If a specified item contains a space, it must be enclosed in quotes such as `"/home/user/music/my song.mp3"`.

14.1.6 LOOKUP

Usage: **LOOKUP** [clustered|unclustered|all]

Implemented: Picard 2.9

Lookup files in the clustering pane. Options are clustered files, unclustered files or all files. If not specified, the command defaults to all files.

This is the equivalent of using the *“Tools → Lookup”* command.

14.1.7 LOOKUP_CD

Usage: **LOOKUP_CD** [device/log file]

Implemented: Picard 2.9

Read CD from the selected drive or ripper log file, and looks it up on MusicBrainz. If no argument is specified, it defaults to the first (alphabetically) available disc drive.

This is the equivalent of using the *“Tools → Lookup CD...”* command.

14.1.8 PAUSE

Usage: **PAUSE** <number of seconds to pause>

Implemented: Picard 2.9

Pause executable command processing for the specified number of seconds.

14.1.9 QUIT

Usage: **QUIT** [force]

Implemented: Picard 2.9

The QUIT command waits until all queued executable commands have completed, and then initiates a shutdown request the same as if the user closed Picard from the user interface. This allows Picard to perform the same checks for unsaved files and such. When 'force' is entered as an argument to the command, Picard will bypass the unsaved files check.

Once a QUIT command has been queued, Picard will not queue any further executable commands. If the user cancels the QUIT from the unsaved files check dialog, the system will allow more commands to be queued.

14.1.10 REMOVE

Usage: **REMOVE** <path to one or more files>

Implemented: Picard 2.9

Removes the specified file(s) from Picard. Does nothing if no arguments provided.

14.1.11 REMOVE_ALL

Usage: **REMOVE_ALL**

Implemented: Picard 2.9

Removes all files from Picard.

14.1.12 REMOVE_EMPTY

Usage: **REMOVE_EMPTY**

Implemented: Picard 2.9

Removes all empty clusters and albums.

14.1.13 REMOVE_SAVED

Usage: **REMOVE_SAVED**

Implemented: Picard 2.9

Removes all saved files from the album pane.

14.1.14 REMOVE_UNCLUSTERED

Usage: **REMOVE_UNCLUSTERED**

Implemented: Picard 2.9

Removes all unclustered files from the cluster pane.

14.1.15 SAVE_MATCHED

Usage: **SAVE_MATCHED**

Implemented: Picard 2.9

Saves all matched files from the album pane.

14.1.16 SAVE_MODIFIED

Usage: **SAVE_MATCHED**

Implemented: Picard 2.9

Saves all modified files from the album pane.

14.1.17 SCAN

Usage: **SCAN**

Implemented: Picard 2.9

Scans all files in the cluster pane. This is the equivalent of using the *“Tools → Scan”* command.

14.1.18 SHOW

Usage: **SHOW**

Implemented: Picard 2.9

Make the running instance of Picard the currently active window.

14.1.19 SUBMIT_FINGERPRINTS

Usage: **SUBMIT_FINGERPRINTS**

Implemented: Picard 2.9

Submits outstanding acoustic fingerprints for all (matched) files in the album pane. This is the equivalent of using the *“Tools → Submit AcoustIDs”* command.

14.1.20 WRITE_LOGS

Usage: **WRITE_LOGS <path to output file>**

Implemented: Picard 2.9

Writes the Picard logs to the specified output file. This is the equivalent of using the *Save As...* button from the log viewing screen opened using the *“Help → View Error/Debug Log”* command.

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.

15.1 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](#) that have been submitted to the MusicBrainz Picard repository shown on the Picard website.

15.2 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](#), and there is a thread specific to [file naming and script snippets](#).

15.2.1 File Naming Script

Multiple file naming scripts can be defined in a user’s settings, although only one is selected at a time for use. File naming scripts 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.

File naming scripts are managed using the [File Naming Script Editor](#) which can be opened from the “File Naming” section of the “*Options → Options...*” menu, or directly from the “*Options → Open file naming script editor...*” menu item. The current file naming script can also be selected directly from the “*Options → Select file naming script*” menu.

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

15.2.2 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.

15.2.3 Tagging Script Examples

The following scripting examples show how tagger scripts can be used to solve some specific use cases. Please refer to [Picard scripting language](#) for a detailed description of the variables and functions used in these examples.

Move disambiguation to album title

Append the disambiguation comment of a release to the album title:

```
$set(album,%album%$if(%_releasecomment%, \(%_releasecomment%\)))
```

Release language as language

The `%_releaselanguage%` variable specifies the language of the track listing, whereas the `%language%` variable is supposed to be the lyrics language. The following script will use the `%_releaselanguage%` instead if `%language%` is empty:

```
$if($not(%language%),$set(language,%_releaselanguage%))
```

Use original release date

By default Picard provides a tag `date` which holds the release date of a specific release and `originaldate` which provides the earliest release date of this release. For example you might have a 2020 reissue of an album that originally was released in 1992. In this case `date` will be set to “2020” and `originaldate` to “1992”. If you prefer to have always the original release date as the primary date in your file’s tags you could use the following script:

```
$set(date,$if2(%originaldate%,%date%))
```

The use of `$if2` ensures that if `originaldate` is empty it will fall back to `date`.

In addition Picard provides a variable `%_recording_firstreleasedate%`, which tries to provide the first release date per recording (which can be different for each track in a release). If you prefer this you can use the following script:

```
$set(date,$if2(%_recording_firstreleasedate%,%originaldate%,%date%))
```

Or if you want to keep the date for the actual release date of the specific release, but use the recording’s first release date as `originaldate`:

```
$set(originaldate,$if2(%_recording_firstreleasedate%,%originaldate%))
```

Set album sort name

The `albumsort` tag is not filled by Picard by default. You can set it to a meaningful value with prefixes “The” and “A” moved to the end with the following script:

```
$set(albumsort,$swapprefix(%album%))
```

This will e.g. set the sort name for the release “The Best of Muddy Waters” to “Best of Muddy Waters, The”.

Set compilation for multi artist releases

By default the compilation tag will be set to 1 only for Various Artists releases. The following script will set it for all releases with more than one artist (as it was default behavior in Picard 1.2 and earlier):

```
$if(%_multiartist%, $set(compilation,1))
```

Remove featuring from album artist

This always removes featuring artists from the album artist:

```
$set(albumartist,$replace(%albumartist%,\\s+feat\\..*,))
```

Move featuring from artist to title

According to MusicBrainz guidelines featuring artists are part of the artist name, e.g. "Artist A feat. Artist B". Some users prefer to have featuring added to the album or track title instead. The following script moves featured track artists to the track title:

```
$set(_feat_title,$rsearch(%artist%,\\s+\\\\(?(f\\(ea\\)?t\\\\.[^\\)]*\\\\))
$set(artist,$rreplace(%artist%,\\s+\\\\(?(f\\(ea\\)?t\\\\.[^\\)]*\\\\)?,))
$set(title,$if(%_feat_title%,%title% \\(%_feat_title%),%title%))
```

The same can be done for moving featured artists from the album artist to the album title:

```
$set(_feat_album,$rsearch(%albumartist%,\\s+\\\\(?(f\\(ea\\)?t\\\\.[^\\)]*\\
$set(albumartist,$rreplace(%albumartist%,\\s+\\\\(?(f\\(ea\\)?t\\\\.[^\\)]*\\\\
$set(album,$if(%_feat_album%,%album% \\(%_feat_album%\\),%album%))
```


Preserve original filename

The `originalfilename` tag is supposed to hold the filename the file originally had. By default Picard does not set or modify this tag. If you want to save this information the following Script can be used:

```
$set(originalfilename,$if2(%originalfilename%,%_filename%.%_extension  
→%))
```

This will keep any existing `originalfilename` tag. But if this tag is not yet present the tag will be set to the current filename. As this happens before the file is being saved the original name of the file before Picard modifies it can be preserved.

15.3 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.

15.3.1 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

15.3.2 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.

15.3.3 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.

15.3.4 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.

15.3.5 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.

15.3.6 Removing Albums

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

15.3.7 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

Sometimes things don't go as planned, and you need to find out what has gone wrong in order to correct the problem. This section provides information on how to get started troubleshooting problems encountered while using MusicBrainz Picard.

16.1 General Troubleshooting

16.1.1 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](#).
- Read the [FAQ section](#) for common questions and problems.
- Consult the [community forums](#).
- Check the [download page](#) for a newer version of Picard which might solve your problem.
- If the problem is to do with a plugin, check the [Picard Plugins](#) for updated plugin versions.

16.1.2 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](#). 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](#) 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?
- The **"Debug"** level log from the Picard session demonstrating the problem.

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.

16.1.3 Getting a Debug Log

For many bugs, it helps developers to have a debug log from Picard. You can see the log by going to *"Help → View Log"*. You can also get a full debug log, which is better because it contains more detailed information. Pasting this log into your forum post or bug ticket can help developers and other users to resolve your issue more quickly. To retrieve the full debug log:

1. Start Picard.
2. Open the log view with *"Help → View Log"*.
3. Change the log level *verbosity* to **Debug**.
4. Close the log viewer.
5. Close and restart Picard.
6. Repeat the action that caused the problem being reported.
7. Open the log viewer and copy the output to paste into the forum post or bug ticket. Alternately, you can save the log to a file to attach to your bug report by using the *Save As...* button.
8. Close the log viewer, and close Picard.

16.1.4 Getting Logs in Case of Crashes

In some cases the problem will cause Picard to crash and not allow you to access the resulting log from the log viewer. You can still generate a log output to attach to your report by starting Picard with the `--debug` command line option from a command / terminal window and copying the log output information from the terminal. The steps to follow for each of the supported platforms are:

Windows Systems

First open a command window by clicking the search icon on the Windows Taskbar and enter “cmd”. Then start Picard by entering the following in the command window:

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

This will display all log entries in the command window, and allow you to copy the information to the clipboard to paste into your report.

Note: This method will only work with the installed version of Picard. It will not work with the portable or Windows Store versions.

macOS Systems

First open a terminal window by doing one of the following:

- Click the Launchpad icon in the Dock, type “Terminal” in the search field, then click *Terminal*.
- In the Finder, open the “/Applications/Utilities” folder, then double-click “Terminal”.

Assuming Picard was put into the system wide Applications folder when installed, it can then be started by entering the following in the terminal window:

```
"/Applications/MusicBrainz Picard.app/Contents/MacOS/picard-run" --  
↪ debug
```

This will display all log entries in the terminal window, and allow you to copy the information to the clipboard to paste into your report.

Linux Systems

First open a Terminal window in your desktop environment, either from the Applications menu or by pressing Ctrl+Alt+T on most systems. Then start Picard by entering the following in the terminal window:

```
picard --debug
```

This will display all log entries in the terminal window, and allow you to copy the information to the clipboard to paste into your report.

16.2 Picard won't start

If you find that Picard won't start there are a few common possible reasons, and things to try to correct the issue. Before doing anything drastic, it is recommended that you try to start Picard from the command line with the `-d` option to generate the debug logging. This process is described in the [General Troubleshooting](#) section. If the resulting logs don't provide any clues as to the problem, it may be one of the following:

The program files have become corrupted

If you suspect that this may be the problem, the first (and simplest) thing to try is to reinstall the program. This should address any potential file corruption issues. If Picard still won't start then it is unlikely that this was the problem.

A plugin file has become corrupted or is incompatible

To check whether one of the plugin files has become corrupted or, in the case of a recent upgrade to a plugin or Picard, a plugin is not compatible, you should try removing all of the plugins and then start Picard. Since you won't be able to disable or remove the plugins using Picard's 'Option' settings, you will need to remove them manually. The plugins may be located in a `plugins` subdirectory of the directory where the Picard program file is stored, or in a user-specific directory:

- *Windows:* C:\Users\user\AppData\Local\MusicBrainz\Picard\plugins
- *macOS:* ~/Library/Preferences/MusicBrainz/Picard/plugins
- *Linux:* ~/.config/MusicBrainz/Picard/plugins

Once you have located the plugin files, they should be removed from the `plugins` directory and moved to a temporary directory. Then try to start Picard. If the program starts, you should try restoring the plugin files from your temporary directory one at a time, and check if Picard will start. This will help identify the plugin that was causing the problem.

The option settings file has become corrupted or is incompatible

To check whether Picard's option settings file has become corrupted or, in the case of a recent upgrade to Picard, it is not compatible, you should try removing the settings file and then start Picard. If Picard is started without finding its configuration settings file, it will create a new one using the default settings. The settings file is called `Picard.ini` and can be found in a user-specific directory:

- *Windows:* C:\Users\user\AppData\Roaming\MusicBrainz
- *macOS:* ~/Library/Preferences/MusicBrainz
- *Linux:* ~/.config/MusicBrainz

Again, it is recommended that you move the file to a temporary directory so that it can be recovered if this turns out not to be the cause of the problem.

There really is a bug in Picard

If this problem started just after updating Picard, in spite of all the testing that is performed prior to releasing a new version, it may be possible that this is indeed a bug. In that case, you should first try to reinstall the previous version to ensure that it works and that the problem is only occurring with the new version. Then you should report the issue, following the steps outlined in the “Reporting a Bug” topic of the [General Troubleshooting](#) section. Please be sure to include as much information as possible, which will help the developers to locate and fix the problem.

16.3 There is no coverart

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

16.3.1 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 “Allowed cover art URLs” 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.

16.3.2 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.

See also:

For more information please see: [AAC Files](#) / [AC3 Files](#) / [ID3 Files](#) / [WAVE Files](#)

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.

16.4 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. Not all file types support all the tags Picard supports.

Please see the [Appendix B: Tag Mapping](#) section for details about the tags supported by various file formats.

16.5 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 if 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.

16.6 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.

16.7 macOS shows the app is damaged

On macOS 10.12 and 10.13 there have been reports that sometimes the MusicBrainz Picard app cannot be started and macOS shows an error message:

“MusicBrainz Picard.app” is damaged and can’t be opened. You should move it to the Trash.

This mostly seems to happen after moving the file to the Applications folder and seems to be caused by Gatekeeper mistakenly marking the app as damaged. To solve the issue open a terminal and run:

```
xattr -c "/Applications/MusicBrainz Picard.app"
```

This will clear the app being marked as damaged. If you have placed the app in a different location then /Applications adjust the path in the command above accordingly.

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.

17.1 Using Picard

17.1.1 How do I tag files with Picard?

There is a separate section that explains the tagging process. Please see [Using Picard](#) for details.

17.1.2 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.

17.1.3 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](#), supplying evidence of the track listing and as much information as possible to prove it is genuine and it may be accepted again.

17.1.4 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.

17.1.5 macOS shows the app is damaged. How can I run Picard?

On macOS 10.12 and 10.13 there have been reports that sometimes the MusicBrainz Picard app cannot be started and macOS shows an error message:

“MusicBrainz Picard.app” is damaged and can’t be opened. You should move it to the Trash.

This mostly seems to happen after moving the file to the Applications folder and seems to be caused by Gatekeeper mistakenly marking the app as damaged. To solve the issue open a terminal and run:

```
xattr -c "/Applications/MusicBrainz Picard.app"
```

This will clear the app being marked as damaged. If you have placed the app in a different location than /Applications adjust the path in the command above accordingly.

17.1.6 Picard is installed on Linux as a Snap, how can I access removable media?

Picard installed as a Snap is running inside a sandbox and thus it does not have full access to all files and folders on your system. By default Picard has access to your home folder. You can additionally give it access to removable media by running the following command on a terminal:

```
snap connect picard:removable-media
```

17.1.7 On Windows, how do I solve errors on saving to cloud storage drives mounted with rclone?

rclone can provide access to cloud storage by mounting a virtual filesystem as a drive. This virtual filesystem has some differences to a real filesystem which can cause compatibility issues.

For full compatibility with Picard you need to mount the cloud storage with rclone as a network drive with the `--network-mode` parameter and set the cache mode to `--vfs-cache-mode=writes` or `--vfs-cache-mode=full`. Your rclone command to mount a remote as drive X: might look like this:

```
rclone mount --vfs-cache-mode=writes --network-mode remote:path/to/  
→files X:
```

Please refer to the rclone documentation for more details.

17.2 File Formats

17.2.1 What formats does Picard support?

Picard supports the following file formats:

- MPEG-1 Audio (.mp3, .mp2, .m2a)
- MPEG-4 Audio (.m4a, .m4b, .m4p, .m4v, .m4r, .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)
- Generic Ogg files (.ogg, .ogx)
- Ogg FLAC (.ogg, .oga)
- Ogg Theora (.ogg, .ogv)
- Ogg Opus (.opus)
- Ogg Audio (.oga)
- Ogg Video (.ogv)
- ADTS stream / AAC (.aac)
- AC-3 (.ac3, .eac3)
- Direct Stream Digital (.dff, .dsf)

Note: WAVE files lack a standard for proper tagging. Picard uses ID3v2 tags to tag WAVE files, but this is not supported by all software. For compatibility with software which does not support ID3v2 tags in WAVE files additional RIFF INFO tags can be written to the files. RIFF INFO has only limited support for tags and character encodings.

17.2.2 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.

17.2.3 What rippers are supported for looking up from logs?

As of version 2.9, Picard supports the use of log files produced by popular CD file rippers for looking up a release. Because the log files of these rippers contain sufficient information to generate the CD table of contents they can be used in place of reading the CD itself. The supported rippers include:

- [dBpoweramp](#) for macOS and Windows
- [Exact Audio Copy \(EAC\)](#) for Windows
- [fre:ac](#) for Linux, macOS, Windows and others
- [Whipper](#) for Linux
- [X Lossless Decoder \(XLD\)](#) for macOS

17.2.4 Which tags can Picard write to my files?

See the [Tags & Variables](#) section for information on which MusicBrainz fields Picard writes to tags. [Picard Tag Mapping](#) contains more technical information on how these are further mapped into each tag format.

17.2.5 How do I edit tags in several files at once?

1. Click and select several files with Ctrl or Shift.
2. The metadata view at the bottom will show which tags are present in the selected files and whether they are the same across all files or different.
3. If you edit any value in the “New values” column you will change this tag for all selected files.
4. You need to click Save in order to persist these changes to your files.

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](#), [foobar2000](#), 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!

17.2.6 Why is saving files sometimes slow, but saving a second time much faster?

In most file formats the tags are near the beginning of the file, before the actual music data. If changed tags get written to the file and the newly written tags take more space than before the entire file needs to be rewritten. This is usually much slower than just rewriting part of the file containing the tags, especially for larger files and/or if the files are on a slow storage (e.g. a network share or slow external drive).

To mitigate the issue most tagging software (including Picard) leaves some free space (the so called padding) after the tags and before the actual music data. If the newly written are only a bit larger than before this free space can be used instead of rewriting the entire files. Likewise if the newly written tags take less space than before this only leads to an increase in padding, avoiding rewriting the file.

This all means that when you add many tags to the files (or if there is no or only small padding) you experience slow writing speed. If you do only small changes or just remove and later re-add tags the writing is much faster.

17.2.7 Why does Picard not use Vinyl style track numbers (e.g. A1, A2, ...) by default?

For Vinyl releases the track numbers on MusicBrainz are usually entered as A1, A2, ..., B1, B2, ... and so on. Other releases might use even different more uncommon numbering schemes. Yet Picard will by default always write decimal track numbers, starting with 1 for the first track on a medium.

The main reason for this is that this is how track numbers are defined for most file formats. The formats expect decimal numbers, and likewise music players might only expect decimal numbers when reading the files.

If you really want to you can use the scripting variable `%_musicbrainz_tracknumber%` which always holds the track number as it was entered in the MusicBrainz database. The following script will set the tracknumber tag to the value as displayed in the MusicBrainz database:

```
$set(tracknumber,%_musicbrainz_tracknumber%)
```

Please be aware that for MP4 files this will result in the track number not being saved, as the MP4 format does not allow for non integer values in this tag. For other formats it depends on the playback software and devices you use if they can handle these non-standard track numbers.

17.2.8 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](#) 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](#).

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.

17.2.9 I am using Fedora. Why doesn't acoustic fingerprinting work?

Acoustic fingerprinting in Picard uses a tool called **fpca1c**, which is not available in Fedora. You can get it by installing the `chromaprint-tools` package from the [RPM Fusion repository](#). This functionality is not contained in the main Fedora `picard` package because it requires the `ffmpeg` package which [cannot be distributed by Fedora](#). After [enabling the "rpmfusion-free" RPM Fusion repository](#), install the package (as root) using:

```
yum install chromaprint-tools
```

17.3 Configuration

17.3.1 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`

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`

17.3.2 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](#).

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.

17.3.3 My tags are truncated to 30 characters in Windows Media Player!

Picard's default settings write ID3v2.4 and ID3v1 tags to files. Older WMP versions can't read ID3v2.4, so it falls back to ID3v1 which has a limitation of 30 characters per title. To resolve this issue, configure Picard to write ID3v2.3 tags instead.

Since Windows 10 Creators Update (version 1703) ID3v2.4 is supported and the above issue should no longer apply.

17.3.4 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.

TUTORIALS

18.1 Writing a File Naming Script

Writing a script to organize and name your files is actually not that hard – just don’t get intimidated by all the ‘\$’, ‘%’ and parentheses. If you can write down a pattern like “**ARTIST - (YEAR) ALBUM NAME/TRACK - SONG TITLE**” of how you want the files and folders named, you can quite easily translate this to the proper script.

To get started, first open the *File Naming Script Editor*, either by selecting “Options → Open file naming script editor...” from Picard’s main menu bar or by clicking the *Edit script...* button on the *File Naming Options* configuration page. From this screen, you can start a new script for your work.

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. In our example, we only have one ‘/’ character, meaning that we will have one directory level for the album which will contain the songs for that album.

First, let’s have a look at what we need. You see a list of the available tags in the *Basic Tags* section. We want the **ARTIST** name, so available tags for this could be `albumartist` or `artist`. This should be the name for an album folder, so `albumartist` sounds like what we need. To get the actual value for a tag you need to enclose its name in percent signs. So let’s start:

```
%albumartist%
```

Now we want the **YEAR**. There is no year tag, but there is date. Let’s use this for now. If we want to add extra text like the “-” just write it down. We need to be careful with the parentheses, because those are special variables in scripting. We need to prefix them with a backslash. Let’s add this all:

```
%albumartist% - \(%date%\)
```

Now we want the **ALBUM NAME**. That’s simple, just use `album`:

```
%albumartist% - \(%date%\) %album%
```

That takes care of the directory portion of the renaming. The next part is the **TRACK** number and **SONG TITLE**. The track number is available as `tracknumber` and the title of the track is simply `title`. Adding these to our script we get:

```
%albumartist% - \(%date%\) %album%/ %tracknumber% - %title%
```

You can see that this looks nearly like the pattern that we said we wanted at the start. It's not perfect yet for a few reasons. What if there are 10 or more tracks on the album and they don't sort properly in the directory listing? Also, we get a full date instead of just the year. Finally, sometimes if you tag existing files they might not have the `albumartist` set, just `artist`.

Let's fix the track number first. We can take care of that by using the `$num` function to add a leading zero to the number shown for tracks 1 through 9:

```
%albumartist% - \(%date%\) %album%/ $num(%tracknumber%,2) - %title%
```

Now let's fix the **ARTIST**. We can fallback to using `artist` if `albumartist` is not available by using:

```
$if2(%albumartist%,%artist%) - \(%date%\) %album%/ $num(%tracknumber%,  
→2) - %title%
```

The `$if2` function uses the first value that is not empty, so if `albumartist` is empty it uses `artist` instead.

For the date tag the dates from MusicBrainz are always formatted as YYYY-MM-DD. We only need the year, so let's get just the first 4 characters with the `$left` function:

```
$if2(%albumartist%,%artist%) - \($left(%date%,4)\) %album%/ $num(  
→%tracknumber%,2) - %title%
```

What happens if there is no date tag information? Sometimes MusicBrainz does not have the release date of an album set as it is not yet known or hasn't been entered into the database. It would be great to omit the entire date with the parentheses in this case. Let's use the `$if` function to check whether the date is set:

```
$if2(%albumartist%,%artist%) - $if(%date%,\($left(%date%,4)\) )%album%/  
→$num(%tracknumber%,2) - %title%
```

Alternately, we can enter a placeholder such as "0000" if the date is missing:

```
$if2(%albumartist%,%artist%) - \($if(%date%, $left(%date%,4),0000)\)  
→%album%/ $num(%tracknumber%,2) - %title%
```

And there you have it – the final script for naming your files developed from the pattern that we used as our starting point.

See also:

For additional information about the available tags and variables please see the [Tags & Variables](#) section. For information about the script functions available please see the [Scripting Functions](#) section.

18.2 Understanding Acoustic Fingerprinting and AcoustIDs

The fingerprinting is the basis for the whole AcoustID song identification system. The audio fingerprint captures the characteristics of the recording, but there can be slight differences in the fingerprint of files of the same recording caused by such things as different encoding or bitrate. Fingerprints, along with the track metadata, are submitted to the AcoustID website where the AcoustID server combines fingerprints that are similar enough and assigns them a single AcoustID. This is actually what makes the AcoustID system really work for audio identification. The same recording can generate many slightly different fingerprints, but the AcoustID represents what the service identifies as the same recording for all of the associated fingerprints.

What Picard does is as follows:

1. When you click “Scan” on a file, Picard generates the audio fingerprint for the file, using the **fpCALC** command line utility provided by AcoustID.
2. Picard uses this fingerprint to lookup an AcoustID from the AcoustID server. The AcoustID server will compare the fingerprint and will try to match it to an existing AcoustID. There are three possibilities:
 - It doesn’t find an AcoustID. The lookup failed.
 - The AcoustID server finds an existing AcoustID for the submitted fingerprint, but it is not associated with any MusicBrainz recording. The lookup failed.
 - The AcoustID server finds an existing AcoustID for the submitted fingerprint and it is associated with a MusicBrainz recording. Picard matches the file to one of the MusicBrainz recordings linked to the AcoustID.

If there was no AcoustID found you can use the “Submit” button in Picard to submit the fingerprints to the AcoustID server once you have matched the files to the proper recordings. If there is no AcoustID already existing for a fingerprint, the server will generate a new AcoustID (which can take some time). It will also link the AcoustID to the MusicBrainz recording identified by the submitted metadata. Please see the [Submitting Acoustic Fingerprints](#) section for a detailed step-by-step procedure.

You don’t need the AcoustID fingerprinting software to manually generate new AcoustIDs. The difference is, that the fingerprinting software is meant to be run on already tagged files, so if it cannot find an AcoustID it will immediately do the submission. For Picard the AcoustID is primarily an identification tool, and because the files are considered untagged at this *identification* stage, you can only do the submission once the files have been properly matched to a MusicBrainz recording. You will also find that after submission Picard will not automatically fetch the newly generated AcoustIDs. This is because the generation can take some time, and the response received from the AcoustID server does not contain newly generated AcoustIDs. However, if you do another scan on the files after submission, the AcoustID should be available.

Note: If files are matched using “*Scan*” and then “*Generate fingerprints*” is used on them, submission will not be enabled, because they have already been matched by fingerprint. This is the same situation as just using “*Scan*”, because after the files are scanned the resulting fingerprint / recording ID is remembered as already having been submitted.

Also if you have files matched to tracks and use “*Generate fingerprints*” and are able to successfully submit the fingerprints, attempting to use “*Generate fingerprints*” for the same files and tracks again does not re-enable submission for those files. The reason is the same: Picard remembers the fingerprint / recording ID combinations already being submitted. However, restarting Picard (or even just removing and re-adding those files) and then using “*Generate fingerprints*” will enable submission again.

Fingerprints are submitted in batches depending on fingerprint size, but often up to 200 or 250 fingerprints can be submitted in one batch. A submission request for a batch might fail due to various reasons such as networking or server issues. If a request fails, all of the fingerprints of this submission batch are still marked as not having been submitted and submission could be retried.

You can also use “*Generate fingerprints*” on either unmatched or matched files. This will only generate the acoustic fingerprints without doing any lookup on the AcoustID server. This also means there will be no AcoustID tag created. However, you can submit these fingerprints if you match the fingerprinted files to a track.

18.3 Handling of multiple release countries

Some releases, especially digital releases, can have a very long list of release countries, sometimes listing all of the world's countries except for a few where the release is not officially available. Picard offers some tools to handle this.

Let's take the release **Bleach**, by Nirvana (MusicBrainz release [adab3feb-1822-4d27-a997-db7d6c9688c0](#)) as an example.

By default Picard will write a single `releasecountry` tag to the files. Prior to v2.3.1, Picard had been populating the tag with what the MusicBrainz server returned as the country for the release. If there were multiple release events, this country field was just filled with the first one in alphabetical order (Afghanistan in our example). Picard v2.3.1 introduced some options to better handle this.

18.3.1 Using preferred release countries

If you configure preferred release countries in “*Options → Metadata → Preferred Releases*”. Picard will use the first country from the preferred release countries that is also in the list of release events. So if you have configured preferred release countries to be Europe, Canada, Germany and UK, for our example that would mean the `releasecountry` tag gets set to Canada.

18.3.2 Using scripting to set a different country

Picard v2.3.1 also added a new variable `%_releasecountries%`, which provides the complete list of release countries for a release as a multi-value variable. You can use this to set different values for the `releasecountry` tag.

For example, the following script would set it to “[International]” if there are 10 or more release countries:

```
$if($gte($lenmulti(%_releasecountries%),10),$set(releasecountry,
↪[International]))
```

Of course you can adjust the count and the replacement text to your liking. You can also choose to save the entire list instead of just a single country to this tag using the script:

```
$setmulti(releasecountry,%_releasecountries%)
```

Perhaps you prefer to limit this list to the first few entries. The following example just uses the first 6 countries:

```
$setmulti(releasecountry,$slice(%_releasecountries%,0,6))
```


18.3.3 What's missing?

Countries are currently written to the tags as their ISO 3166-1 country code, with some special values added for historical countries and things like [Europe] or [Worldwide]. These codes are not always easily recognizable or obvious, such as “DZ” for Algeria or “DE” for Germany. You can of course use scripting to make these more readable. For example, if you want to see “United Kingdom” instead of “GB” in this tag use:

```
$if($eq(%releasecountry%,GB), $set(releasecountry,United Kingdom))
```

This might work if you deal only with a couple of countries in your collection, or you just want to handle some special cases like using “Europe” instead of “XE” such as in the following script:

```
$if($eq(%releasecountry%,XE), $set(releasecountry,Europe))  
$if($eq(%releasecountry%,XU), $set(releasecountry,[Unknown]))  
$if($eq(%releasecountry%,XW), $set(releasecountry,[Worldwide]))  
$if($eq(%releasecountry%,XG), $set(releasecountry,DDR))
```

A simpler method would be to use the `$countryname()` function introduced in Picard v2.7 to easily convert the code into a readable name, such as in the following scripts:

```
$noop( Convert only %releasecountry% )  
$set(releasecountry, $countryname(%releasecountry%,yes))  
  
$noop( List all countries by name )  
$setmulti(releasecountry, $map(%_releasecountries%, $countryname(%_loop_  
→value%,yes)))  
  
$noop( List only the first 6 countries by name )  
$setmulti(temp, $slice(%_releasecountries%,0,6))  
$setmulti(releasecountry, $map(%temp%, $countryname(%_loop_value%,yes)))
```

18.4 Writing a Plugin

You have a great idea for extending Picard with a plugin but don't know where to start. Unfortunately, this is a common problem and prevents far too many of those great ideas from ever seeing the light of day. Perhaps this tutorial will help get you started in turning your great idea a reality.

Picard plugins are written in Python, so that's the programming language you'll be using. Please check the [INSTALL.md](#) file in the Picard repository on GitHub to see the minimum version requirements. This is Python 3.6 as of the time this tutorial was written. Also refer to the [Plugins API](#) for additional information, including the parameters passed to each of the function types.

For the purpose of this tutorial, we're going to develop a simple plugin to save the argument information provided by Picard to track and release processing plugins. This will demonstrate how the information is accessed, and will provide a utility that you might find useful when developing your own plugins.

The first thing that we'll need to include is the header information that describes the plugin.

```
PLUGIN_NAME = "Example plugin"
PLUGIN_AUTHOR = "This authors name"
PLUGIN_DESCRIPTION = "This plugin is an example"
PLUGIN_VERSION = '0.1'
PLUGIN_API_VERSIONS = ['2.2']
PLUGIN_LICENSE = "GPL-2.0-or-later"
PLUGIN_LICENSE_URL = "https://www.gnu.org/licenses/gpl-2.0.html"
```

Next we list the modules that will be referenced in our code. In this case, we will be using the `os` module to build the output file path, and the `json` module to format the argument dictionary text for readability. We will be saving our output file to the base directory used for file naming so we import the `config` module from Picard, as well as the `log` module so that we can write debug or error messages to Picard's log. Finally, we import the appropriate processing hooks and plugin priority settings.

```
import json
import os

from picard import config, log
from picard.metadata import (register_album_metadata_processor,
                             register_track_metadata_processor)
from picard.plugin import PluginPriority
```

Warning: To ensure maximum compatibility, you should only use standard Python modules, or third-party modules that are already included in Picard. If you use other modules, then the plugin will not function properly if used on a system that doesn't

have the proper version of the module installed or if someone is using an executable version of Picard.

Now we can start adding the code that we want Picard to execute. First we'll identify the output file to store the parameter information provided by Picard. This is a file named `data_dump.txt` to be stored in the file naming output directory. We find the name of the configuration setting we need, `move_files_to`, by examining the Picard source code for the corresponding option setting screen. In this case it is a `TextOption` in the `RenamingOptionsPage` class found in the file `picard/ui/options/renaming.py`.

```
file_to_write = os.path.join(config.setting["move_files_to"], "data_
→dump.txt")
```

The next part is a function to write a Python object to our output file. To allow the same function to be used for different situations, we include parameters to identify the type of line (input type), the object to write, and options for writing to JSON format and appending or overwriting an existing output file. In our case, we want to overwrite the file each time a new release is processed, but always append the track information to the file.

We also include error checking to write an entry to the Picard log in the event of an exception.

```
def write_line(line_type, object_to_write, dump_json=False,
→append=True):
    file_mode = 'a' if append else 'w'
    try:
        with open(file_to_write, file_mode, encoding="UTF-8") as f:
            if dump_json:
                f.write('{0} JSON dump follows:\n'.format(line_type,))
                f.write('{0}\n\n'.format(json.dumps(object_to_write,
→indent=4)))
            else:
                f.write("{0:s}: {1:s}\n".format(line_type, str(object_
→to_write),))
    except Exception as ex:
        log.error("{0}: Error: {1}".format(PLUGIN_NAME, ex,))
```

Now we include the functions to be called when releases and tracks are retrieved by Picard. The release function hook provides three arguments, and the track function hook provides four arguments. The argument types are described in the [Plugins API](#) section. The first argument, `album`, is an object that holds information about the selected album. See the `Album` class in the `picard/album.py` file in Picard's source code for more information.

The second argument, `metadata`, is an object that holds the tags and variables that Picard has assigned for the current release and track. This is where you can add or edit the tags and variables that Picard makes available to the user for scripts. See

the Metadata class in the `picard/metadata.py` file in Picard's source code for more information.

The track and release arguments are Python dictionaries containing the information provided in response to Picard's calls to the MusicBrainz API. The information may differ, depending on the user's *Metadata Options* settings for things like "Use release relationships" or "Use track relationships".

```
def dump_release_info(album, metadata, release):
    write_line('Release Argument 1 (album)', album, append=False)
    write_line('Release Argument 3 (release)', release, dump_json=True)

def dump_track_info(album, metadata, track, release):
    write_line('Track Argument 1 (album)', album)
    write_line('Track Argument 3 (track)', track, dump_json=True)
    # write_line('Track Argument 4 (release)', release, dump_json=True)
```

Finally, we need to register our functions so that they are processed with the appropriate events. In our case, we set the priority to HIGH so that we output the parameter information as it is received by Picard before any other plugins have an opportunity to modify it.

```
# Register the plugin to run at a HIGH priority so that other plugins
↳ will
# not have an opportunity to modify the contents of the metadata
↳ provided.
register_album_metadata_processor(dump_release_info,
↳ priority=PluginPriority.HIGH)
register_track_metadata_processor(dump_track_info,
↳ priority=PluginPriority.HIGH)
```

The complete plugin code file looks something like:

```
PLUGIN_NAME = "Example plugin"
PLUGIN_AUTHOR = "This authors name"
PLUGIN_DESCRIPTION = "This plugin is an example"
PLUGIN_VERSION = '0.1'
PLUGIN_API_VERSIONS = ['2.2']
PLUGIN_LICENSE = "GPL-2.0-or-later"
PLUGIN_LICENSE_URL = "https://www.gnu.org/licenses/gpl-2.0.html"

import json
import os

from picard import config, log
from picard.metadata import (register_album_metadata_processor,
                             register_track_metadata_processor)
from picard.plugin import PluginPriority
```

(continues on next page)

(continued from previous page)

```

file_to_write = os.path.join(config.setting["move_files_to"], "data_
→dump.txt")

def write_line(line_type, object_to_write, dump_json=False,
→append=True):
    file_mode = 'a' if append else 'w'
    try:
        with open(file_to_write, file_mode, encoding="UTF-8") as f:
            if dump_json:
                f.write('{0} JSON dump follows:\n'.format(line_type,))
                f.write('{0}\n\n'.format(json.dumps(object_to_write,
→indent=4)))
            else:
                f.write("{0:s}: {1:s}\n".format(line_type, str(object_
→to_write),))
    except Exception as ex:
        log.error("{0}: Error: {1}".format(PLUGIN_NAME, ex,))

def dump_release_info(album, metadata, release):
    write_line('Release Argument 1 (album)', album, append=False)
    write_line('Release Argument 3 (release)', release, dump_json=True)

def dump_track_info(album, metadata, track, release):
    write_line('Track Argument 1 (album)', album)
    write_line('Track Argument 3 (track)', track, dump_json=True)
    # write_line('Track Argument 4 (release)', release, dump_json=True)

# Register the plugin to run at a HIGH priority so that other plugins
→will
# not have an opportunity to modify the contents of the metadata
→provided.
register_album_metadata_processor(dump_release_info,
→priority=PluginPriority.HIGH)
register_track_metadata_processor(dump_track_info,
→priority=PluginPriority.HIGH)

```

That's it for our plugin code. Now we need to package it so that we can install it into Picard. If we're going to just use it locally for ourself, the easiest way is to just name the file something like `my_plugin.py`. If there are multiple files, such as plugins that include additional settings screens, then the files should be saved in a directory such as `my_plugin` with the main file named `__init__.py`. The directory is then archived into a `my_plugin.zip` file, with the file name the same as the included directory name. The contents of the archive would show as something like:

```
my_plugin/__init__.py
my_plugin/another_file.py
my_plugin/etc
```

If you've made it this far, congratulations! You've just created your first Picard plugin. Now you have a starting point for turning that great idea into reality.

See also:

Relevant portions of Picard's source code including:

- Option settings modules in [picard/ui/options/](#) for names used to access the settings.
- Album class in the [picard/album.py](#) file.
- Metadata class and metadata processing plugin registration functions in the [picard/metadata.py](#) file.
- PluginPriority class in the [picard/plugin.py](#) file.

18.5 Loading releases with MusicBrainz for Android

If you have an Android phone you can use the MusicBrainz for Android app to search for releases by text search or by barcode and load them into Picard running on your computer.

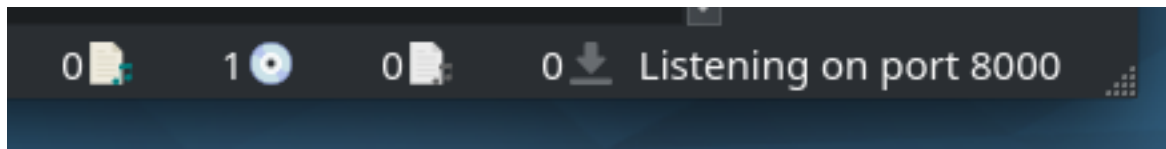
This is useful for example if you have previously ripped your CDs and now you want to tag your ripped files with Picard using exactly the data for your the releases you own. You can then use your phone to scan the barcodes of your CDs and have their data loaded into Picard, then use this data to tag your local files.

For this to work you need both your phone and computer to be connected to the same network.

18.5.1 Configuring Picard

In *Options* → *Options...* → *Advanced* → *Network* enable “Browser Integration” and disable “Listen only on localhost”. It is recommended that you keep the listening port on the default value 8000, but you can change that as well.

Once you have saved the options, check whether Picard is showing a message “Listening on port 8000” in the status bar on the lower right of the main window.



The actual port number can vary, but the default is 8000. Note the port number, you will need it to configure the Android app in the next step.

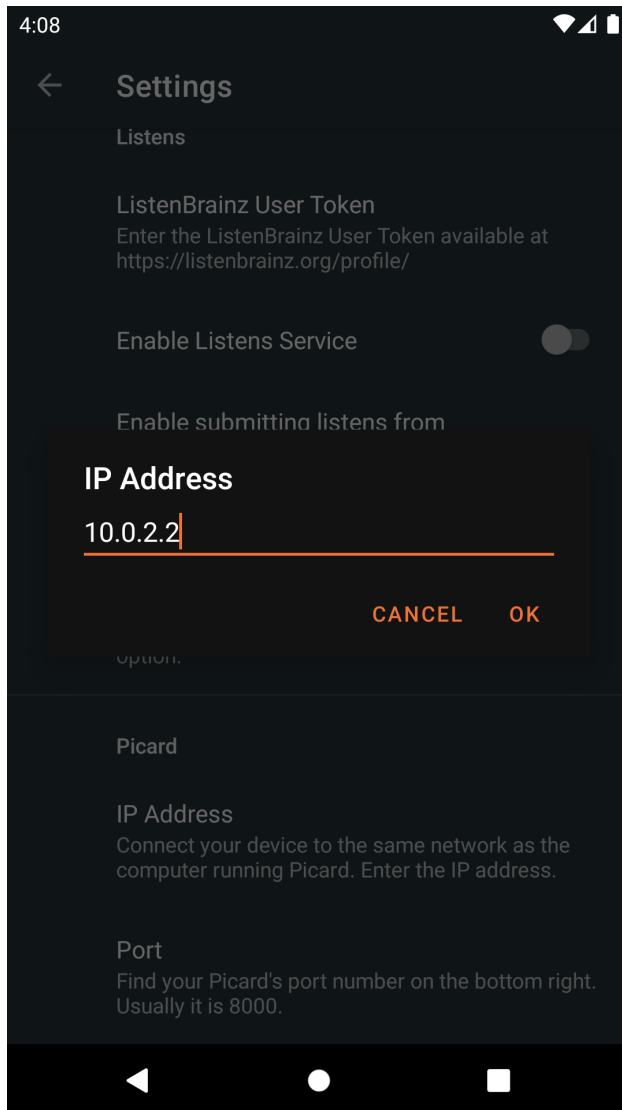
See also:

[Network options](#)

18.5.2 Installing and configuring the MusicBrainz Android app

Install [MusicBrainz for Android](#) on your phone. You can download the latest version of the app either from the [Google Play Store](#) or [F-Droid](#).

Once installed, launch the app and tap on the settings icon on the upper right. Scroll down to the Picard settings. For the IP Address enter the IP address of your computer on which Picard is running. Depending on your local network setup you might also be able to enter the hostname of your computer instead of the IP address.



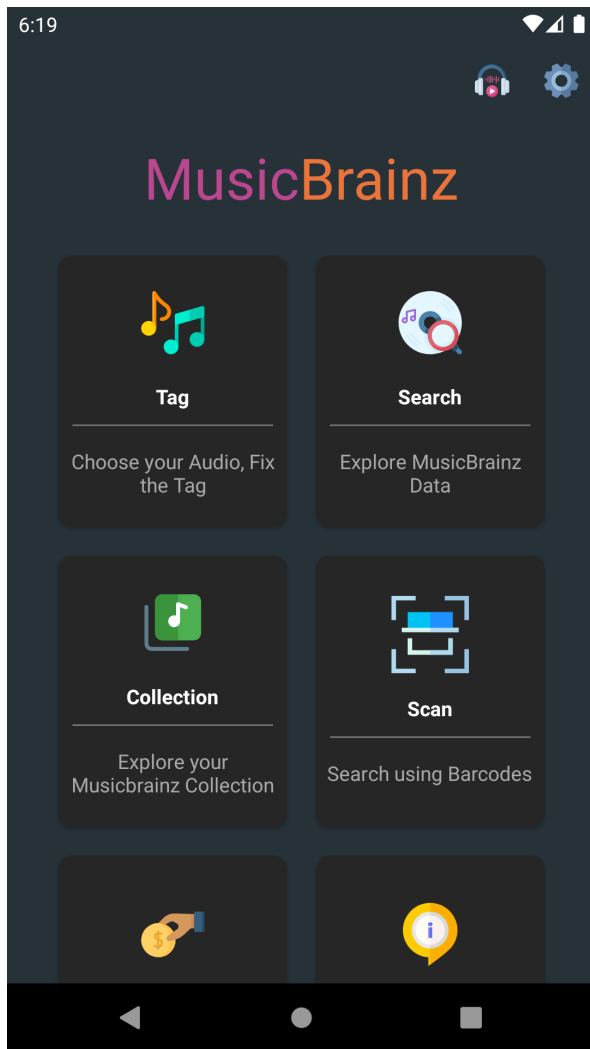
For the Port enter Picard’s listening port as displayed in Picard’s main screen (see the previous section). The default is 8000.

Before you continue make sure Picard is running and the “Listening on port...” status message is shown. Also make sure your phone is connected to your local network.

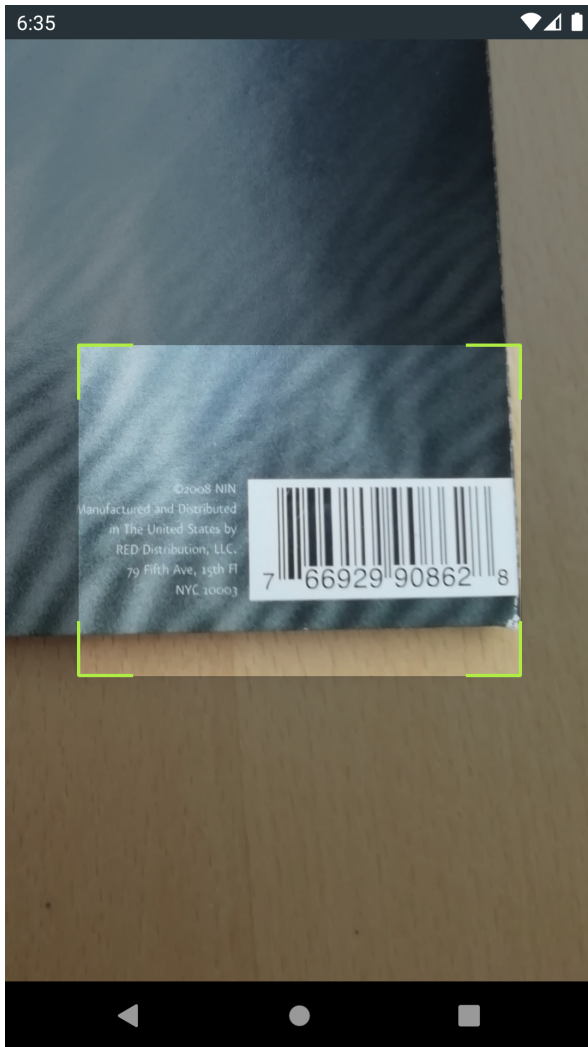
18.5.3 Loading releases by barcode

You can use your phone as a barcode scanner to load the metadata for your physical media:

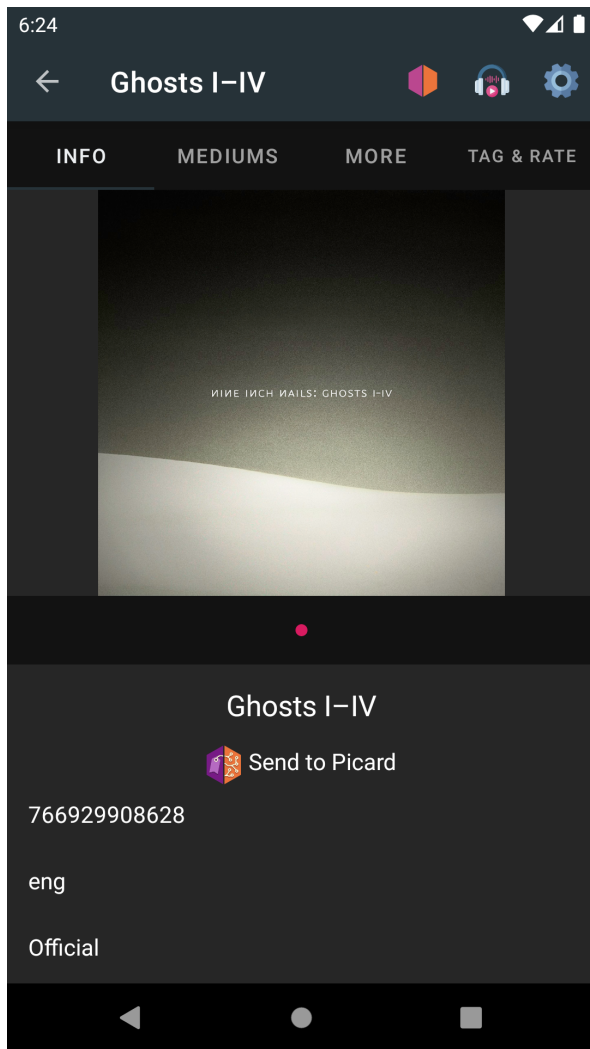
1. On the main screen of the Android app tap on “Scan”.



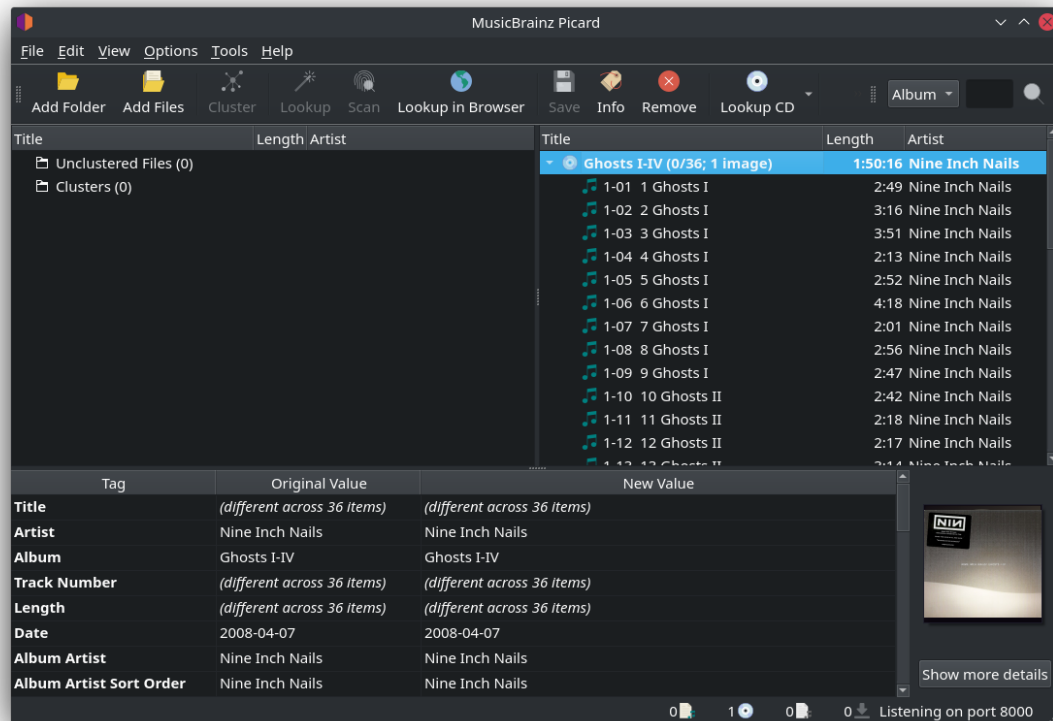
2. Scan the barcode of a CD, LP or other music media.



3. If a release with the scanned barcode is found on MusicBrainz the app will load and show the release details.



4. Tap on “Send to Picard”. If everything was configured correctly the release will be loaded into Picard running on your computer.



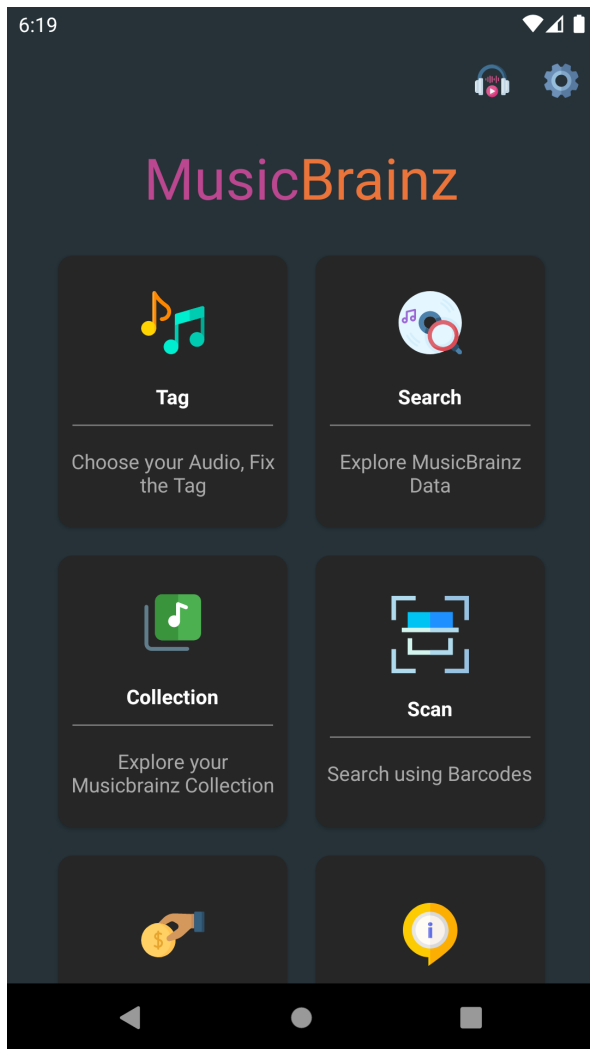
You can now continue tagging your local files by matching them to the loaded tracks as described in [Matching Files to Tracks](#).

Note: If you only want to use the barcode scanner functionality to find and load releases for your physical CDs, LPs or other music media, you can also use the [Picard Barcode Scanner](#) app. The functionality and setup is similar to what is described above, but the app is focused on the barcode scanning and sending the results to Picard.

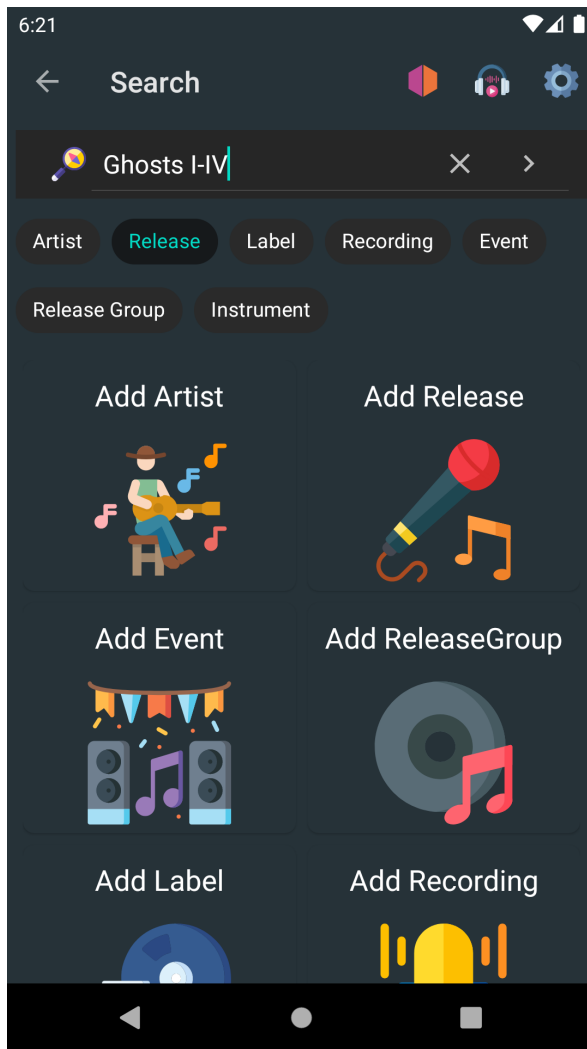
18.5.4 Loading releases by search

Instead of searching by barcode you can also do a text search on your phone:

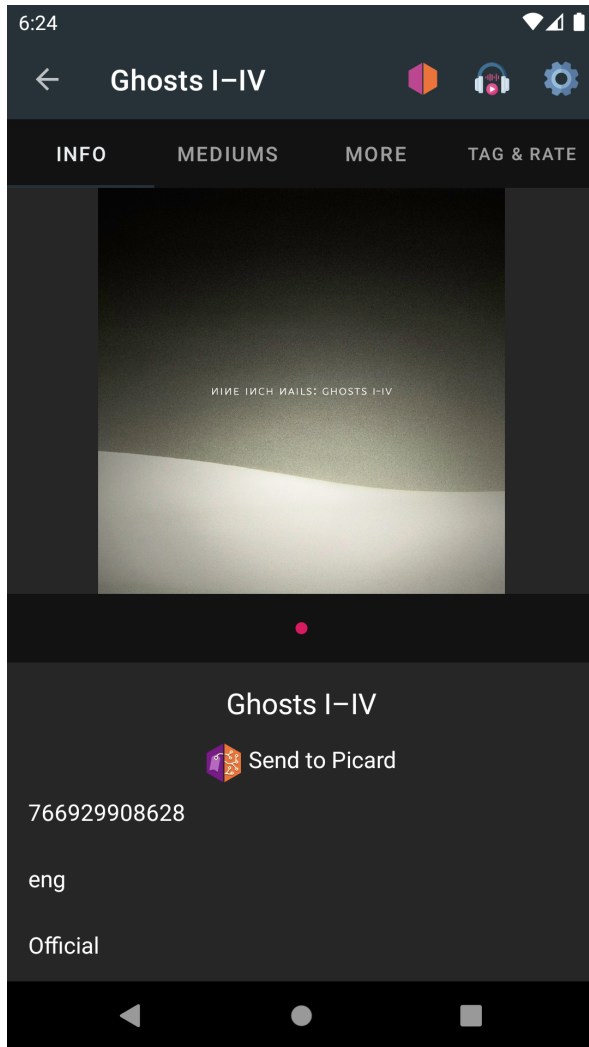
1. On the main screen of the Android app tap on “Search”.



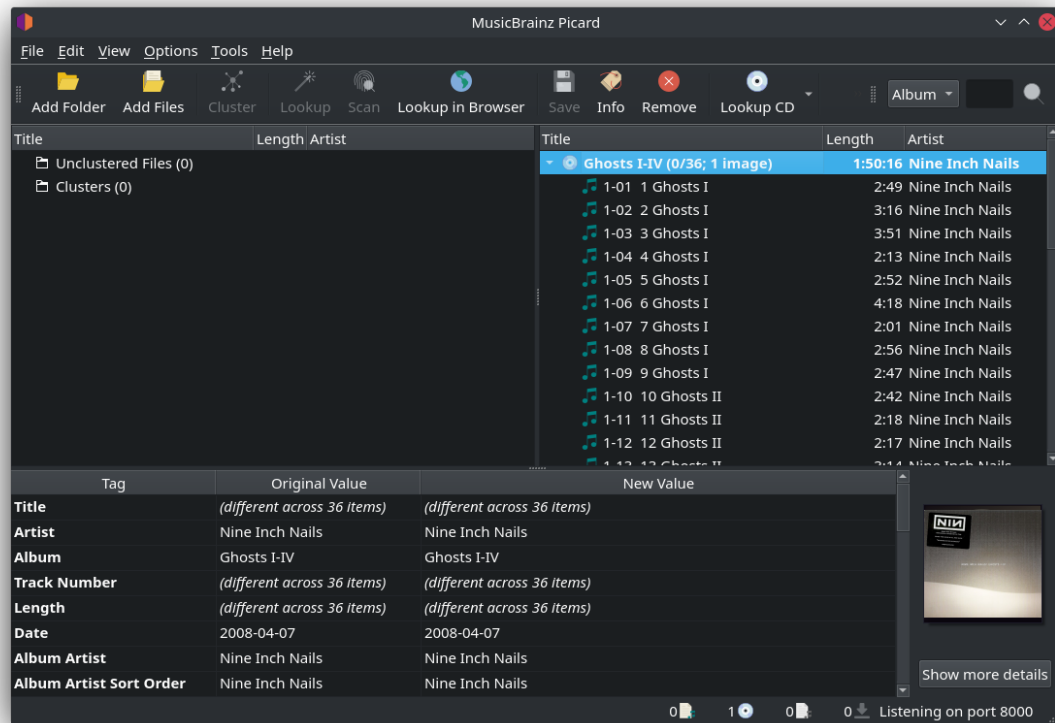
2. On the search page select “Release” and enter a search term, e.g. an album title or artist name.



3. The search results will show a list of matching releases. Tap on one to show the release details.



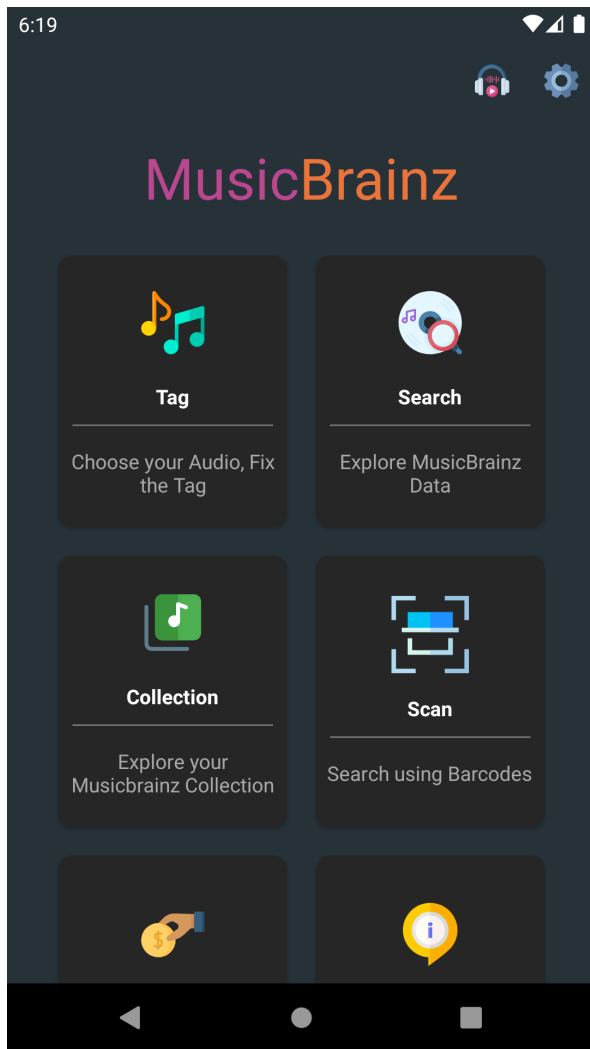
4. Tap on “Send to Picard”. If everything was configured correctly the release will be loaded into Picard running on your computer.



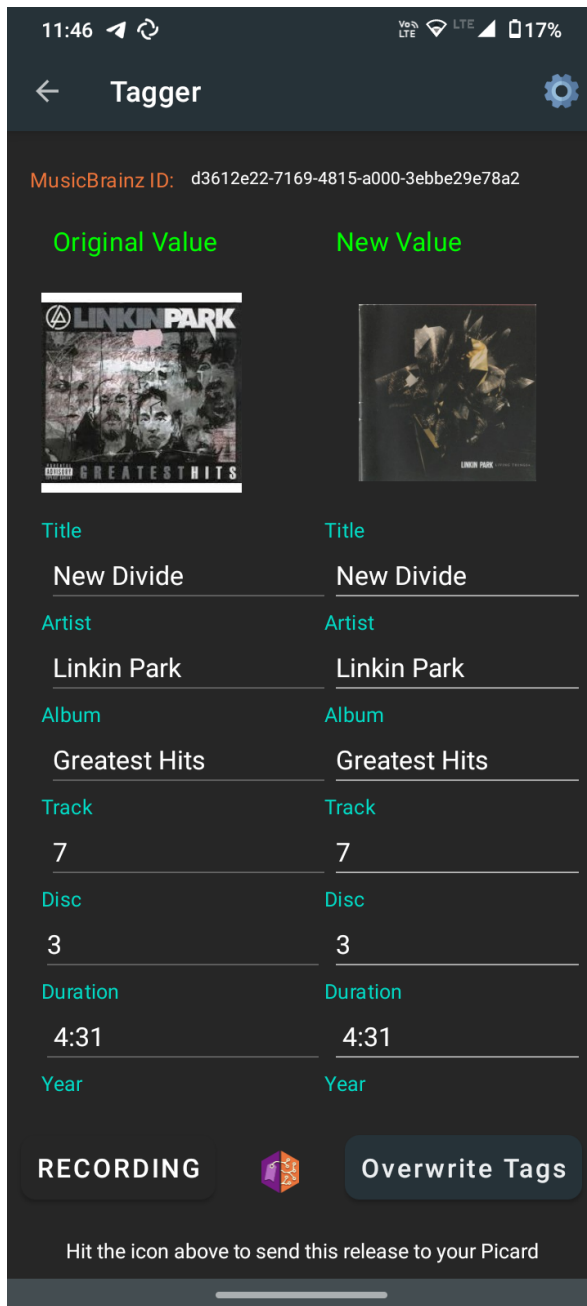
18.5.5 Loading releases from the Tagger

Instead of finding a release by barcode or a search to send to Picard, you can also send a release from a tagged audio file currently stored on your device:

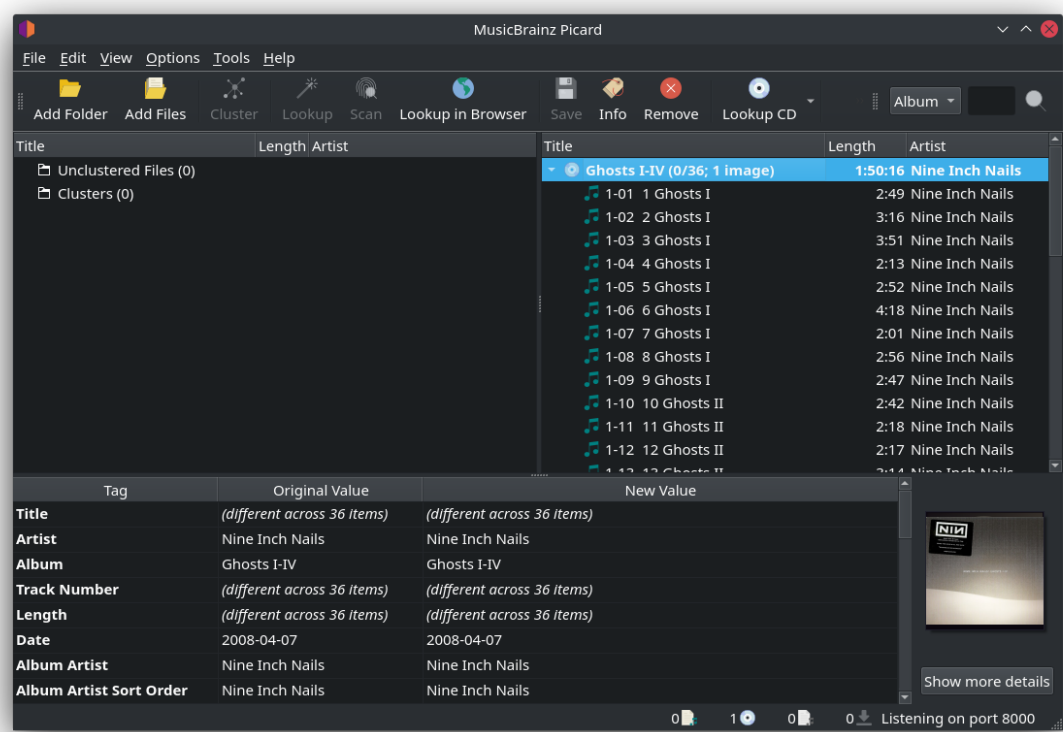
1. On the main screen of the Android app tap on “Tagger”.



2. On the tagger, select your release and tap the MusicBrainz icon near the bottom of the screen.



- If everything was configured correctly, the release will be loaded into Picard running on your computer.



APPENDICES

19.1 Appendix A: Plugins API

19.1.1 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

Since *Picard 2.7* the description can be formatted using
[Markdown](https://daringfireball.net/projects/markdown/) syntax.
If you use Markdown formatting make sure the minimum version in
`PLUGIN_API_VERSIONS` is set to 2.7.
"""
PLUGIN_VERSION = '0.1'
PLUGIN_API_VERSIONS = ['2.7', '2.8']
PLUGIN_LICENSE = "GPL-2.0-or-later"
PLUGIN_LICENSE_URL = "https://www.gnu.org/licenses/gpl-2.0.html"
PLUGIN_USER_GUIDE_URL = "https://my.program.site.org/example_plugin_
↪documentation.html"
```

Variables explanation:

- **PLUGIN_NAME** should be a short but descriptive name for the plugin.
- **PLUGIN_DESCRIPTION** should be as simple as possible, while still describing the main function. If your plugin targets Picard 2.7 or later you can use [Markdown](#) syntax to format the text. If your plugin targets earlier versions you can instead use simple HTML formatting. Please restrict the usage of HTML to basic text formatting (e.g. ``, ``), links (`<a>`) and lists (``, ``).
- **PLUGIN_VERSION** should be filled with the version of Plugin. Plugin versions should be in the format x.y.z (e.g.: "1.0" or "2.12.4"). It is recommended that you use [Semantic Versioning](#).

- **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](#), but you are welcomed to use another license 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.
- **PLUGIN_USER_GUIDE_URL** should be set to a URL pointing to the documentation for the plugin. This variable is optional and may be omitted. If a URL is provided, it will be shown as a clickable link in the description displayed for the plugin in the Plugins option settings screen.

19.1.2 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:

```
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:

```

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)

```

19.1.3 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:

```

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:

```

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).

```
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).

```
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.

```
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.

19.1.4 File Formats

Plugins can extend Picard with support for additional file formats. See the existing [file format implementations](#) for details on how to implement the `_load` and `_save` methods. Example:

```

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"

    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)

```

19.1.5 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)."

```

(continues on next page)

(continued from previous page)

```
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**.

19.1.6 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.
```

(continues on next page)

(continued from previous page)

```
↪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*".

19.2 Appendix B: Tag Mapping

The following is a mapping between Picard internal tag names and those used by various tagging formats. The mapping is also available as a [table](#) and a [spreadsheet](#).

19.2.1 AcoustID

Internal Name	acoustid_id
ID3v2	TXXX:Acoustid Id
Vorbis	ACOUSTID_ID
APEv2	ACOUSTID_ID
iTunes MP4	----:com.apple.iTunes:Acoustid Id
ASF/Windows Media	Acoustid/Id
RIFF INFO	n/a

19.2.2 AcoustID Fingerprint

Internal Name	acoustid_fingerprint
ID3v2	TXXX:Acoustid Fingerprint
Vorbis	ACOUSTID_FINGERPRINT
APEv2	ACOUSTID_FINGERPRINT
iTunes MP4	----:com.apple.iTunes:Acoustid Fingerprint
ASF/Windows Media	Acoustid/Fingerprint
RIFF INFO	n/a

19.2.3 Album

Internal Name	album
ID3v2	TALB
Vorbis	ALBUM
APEv2	Album
iTunes MP4	©alb
ASF/Windows Media	WM/AlbumTitle
RIFF INFO	IPRD

19.2.4 Album Artist

Internal Name	albumartist
ID3v2	TPE2
Vorbis	ALBUMARTIST
APEv2	Album Artist
iTunes MP4	aART
ASF/Windows Media	WM/AlbumArtist
RIFF INFO	n/a

19.2.5 Album Artist Sort Order

Internal Name	albumartistsort
ID3v2	TS02 (Picard>=1.2) TXXX:ALBUMARTISTSORT (Picard<=1.1)
Vorbis	ALBUMARTISTSORT
APEv2	ALBUMARTISTSORT
iTunes MP4	soaa
ASF/Windows Media	WM/AlbumArtistSortOrder
RIFF INFO	n/a

19.2.6 Album Sort Order ^[4]

Internal Name	albumsort
ID3v2	TS0A
Vorbis	ALBUMSORT
APEv2	ALBUMSORT
iTunes MP4	soa1
ASF/Windows Media	WM/AlbumSortOrder
RIFF INFO	n/a

19.2.7 Arranger

Internal Name	arranger
ID3v2	TIPL:arranger (ID3v2.4) IPLS:arranger (ID3v2.3)
Vorbis	ARRANGER
APEv2	Arranger
iTunes MP4	n/a
ASF/Windows Media	n/a
RIFF INFO	n/a

19.2.8 Artist

Internal Name	artist
ID3v2	TPE1
Vorbis	ARTIST
APEv2	Artist
iTunes MP4	©ART
ASF/Windows Media	Author
RIFF INFO	IART

19.2.9 Artist Sort Order

Internal Name	artistsort
ID3v2	TSOP
Vorbis	ARTISTSORT
APEv2	ARTISTSORT
iTunes MP4	soar
ASF/Windows Media	WM/ArtistSortOrder
RIFF INFO	n/a

19.2.10 Artists

Internal Name	artists
ID3v2	TXXX:ARTISTS
Vorbis	ARTISTS
APEv2	Artists
iTunes MP4	----:com.apple.iTunes:ARTISTS
ASF/Windows Media	WM/ARTISTS
RIFF INFO	n/a

19.2.11 ASIN

Internal Name	asin
ID3v2	TXXX:ASIN
Vorbis	ASIN
APEv2	ASIN
iTunes MP4	----:com.apple.iTunes:ASIN
ASF/Windows Media	ASIN
RIFF INFO	n/a

19.2.12 Barcode

Internal Name	barcode
ID3v2	TXXX:BARCODE
Vorbis	BARCODE
APEv2	Barcode
iTunes MP4	----:com.apple.iTunes:BARCODE
ASF/Windows Media	WM/Barcode
RIFF INFO	n/a

19.2.13 BPM ^[4]

Internal Name	bpm
ID3v2	TBPM
Vorbis	BPM
APEv2	BPM
iTunes MP4	tempo
ASF/Windows Media	WM/BeatsPerMinute
RIFF INFO	n/a

19.2.14 Catalog Number

Internal Name	catalognumber
ID3v2	TXXX:CATALOGNUMBER
Vorbis	CATALOGNUMBER
APEv2	CatalogNumber
iTunes MP4	----:com.apple.iTunes:CATALOGNUMBER
ASF/Windows Media	WM/CatalogNo
RIFF INFO	n/a

19.2.15 Comment ^[4]

Internal Name	comment:description
ID3v2	COMM:description
Vorbis	COMMENT
APEv2	Comment
iTunes MP4	©cmt
ASF/Windows Media	Description
RIFF INFO	ICMT

19.2.16 Compilation (iTunes) ^[5]

Internal Name	compilation
ID3v2	TCMP
Vorbis	COMPILATION
APEv2	Compilation
iTunes MP4	cpil
ASF/Windows Media	WM/IsCompilation
RIFF INFO	n/a

19.2.17 Composer

Internal Name	composer
ID3v2	TCOM
Vorbis	COMPOSER
APEv2	Composer
iTunes MP4	©wrt
ASF/Windows Media	WM/Composer
RIFF INFO	IMUS

19.2.18 Composer Sort Order

Internal Name	composersort
ID3v2	TS0C (Picard>=1.3) TXXX:COMPOSERSORT (Picard<=1.2)
Vorbis	COMPOSERSORT
APEv2	COMPOSERSORT
iTunes MP4	soco
ASF/Windows Media	WM/ComposerSortOrder (Picard>=1.3)
RIFF INFO	n/a

19.2.19 Conductor

Internal Name	conductor
ID3v2	TPE3
Vorbis	CONDUCTOR
APEv2	Conductor
iTunes MP4	----:com.apple.iTunes:CONDUCTOR
ASF/Windows Media	WM/Conductor
RIFF INFO	n/a

19.2.20 Copyright ^[4]

Internal Name	copyright
ID3v2	TCOP
Vorbis	COPYRIGHT
APEv2	Copyright
iTunes MP4	cprt
ASF/Windows Media	Copyright
RIFF INFO	ICOP

19.2.21 Date ^[10]

Internal Name	date
ID3v2	TDRC (ID3v2.4) TYER + TDAT (ID3v2.3)
Vorbis	DATE
APEv2	Year
iTunes MP4	©day
ASF/Windows Media	WM/Year
RIFF INFO	ICRD

19.2.22 Director

Internal Name	director
ID3v2	TXXX:DIRECTOR
Vorbis	DIRECTOR
APEv2	Director
iTunes MP4	©dir ^[9]
ASF/Windows Media	WM/Director
RIFF INFO	n/a

19.2.23 Disc Number

Internal Name	discnumber
ID3v2	TP0S
Vorbis	DISCNUMBER
APEv2	Disc
iTunes MP4	disk
ASF/Windows Media	WM/PartOfSet
RIFF INFO	n/a

19.2.24 Disc Subtitle

Internal Name	discsubtitle
ID3v2	TSST (ID3v2.4 only)
Vorbis	DISCSUBTITLE
APEv2	DiscSubtitle
iTunes MP4	----:com.apple.iTunes:DISCSUBTITLE
ASF/Windows Media	WM/SetSubTitle
RIFF INFO	n/a

19.2.25 Encoded By ^[4]

Internal Name	encodedby
ID3v2	TENC
Vorbis	ENCODEDBY
APEv2	EncodedBy
iTunes MP4	©too
ASF/Windows Media	WM/EncodedBy
RIFF INFO	IENC

19.2.26 Encoder Settings ^[4]

Internal Name	encodersettings
ID3v2	TSSE
Vorbis	ENCODERSETTINGS
APEv2	EncoderSettings
iTunes MP4	n/a
ASF/Windows Media	WM/EncodingSettings (Picard>=1.3.1)
RIFF INFO	n/a

19.2.27 Engineer

Internal Name	engineer
ID3v2	TIPL:engineer (ID3v2.4) IPLS:engineer (ID3v2.3)
Vorbis	ENGINEER
APEv2	Engineer
iTunes MP4	----:com.apple.iTunes:ENGINEER
ASF/Windows Media	WM/Engineer
RIFF INFO	IENG

19.2.28 Gapless Playback ^[4]

Internal Name	gapless
ID3v2	n/a
Vorbis	n/a
APEv2	n/a
iTunes MP4	pgap
ASF/Windows Media	n/a
RIFF INFO	n/a

19.2.29 Genre

Internal Name	genre
ID3v2	TCON
Vorbis	GENRE
APEv2	Genre
iTunes MP4	©gen
ASF/Windows Media	WM/Genre
RIFF INFO	IGNR

19.2.30 Grouping ^[3]

Internal Name	grouping
ID3v2	TIT1 GRP1 ^[8]
Vorbis	GROUPING
APEv2	Grouping
iTunes MP4	©grp
ASF/Windows Media	WM/ContentGroupDescription
RIFF INFO	n/a

19.2.31 Initial Key

Internal Name	key (Picard>=1.4)
ID3v2	TKEY
Vorbis	KEY
APEv2	KEY
iTunes MP4	----:com.apple.iTunes:initialkey
ASF/Windows Media	WM/InitialKey
RIFF INFO	n/a

19.2.32 ISRC

Internal Name	isrc
ID3v2	TSRC
Vorbis	ISRC
APEv2	ISRC
iTunes MP4	----:com.apple.iTunes:ISRC
ASF/Windows Media	WM/ISRC
RIFF INFO	n/a

19.2.33 Language

Internal Name	language
ID3v2	TLAN
Vorbis	LANGUAGE
APEv2	Language
iTunes MP4	----:com.apple.iTunes:LANGUAGE
ASF/Windows Media	WM/Language
RIFF INFO	ILNG

19.2.34 License ^[6, 7]

Internal Name	license
ID3v2	WCOP (single URL) TXXX:LICENSE (multiple or non-URL)
Vorbis	LICENSE
APEv2	LICENSE
iTunes MP4	----:com.apple.iTunes:LICENSE
ASF/Windows Media	n/a
RIFF INFO	n/a

19.2.35 Lyricist

Internal Name	lyricist
ID3v2	TEXT
Vorbis	LYRICIST
APEv2	Lyricist
iTunes MP4	----:com.apple.iTunes:LYRICIST
ASF/Windows Media	WM/Writer
RIFF INFO	n/a

19.2.36 Lyrics ^[4]

Internal Name	lyrics:description
ID3v2	USLT:description
Vorbis	LYRICS
APEv2	Lyrics
iTunes MP4	©lyr
ASF/Windows Media	WM/Lyrics
RIFF INFO	n/a

19.2.37 Media

Internal Name	media
ID3v2	TMED
Vorbis	MEDIA
APEv2	Media
iTunes MP4	----:com.apple.iTunes:MEDIA
ASF/Windows Media	WM/Media
RIFF INFO	IMED

19.2.38 Mix-DJ

Internal Name	djmixer
ID3v2	TIPL:DJ-mix (ID3v2.4) IPLS:DJ-mix (ID3v2.3)
Vorbis	DJMIXER
APEv2	DJMixer
iTunes MP4	----:com.apple.iTunes:DJMIXER
ASF/Windows Media	WM/DJMixer
RIFF INFO	n/a

19.2.39 Mixer

Internal Name	mixer
ID3v2	TIPL:mix (ID3v2.4) IPLS:mix (ID3v2.3)
Vorbis	MIXER
APEv2	Mixer
iTunes MP4	----:com.apple.iTunes:MIXER
ASF/Windows Media	WM/Mixer
RIFF INFO	n/a

19.2.40 Mood ^[3]

Internal Name	mood
ID3v2	TM00 (ID3v2.4 only)
Vorbis	M00D
APEv2	Mood
iTunes MP4	----:com.apple.iTunes:M00D
ASF/Windows Media	WM/Mood
RIFF INFO	n/a

19.2.41 Movement ^[4]

Internal Name	movement (Picard>=2.1)
ID3v2	MVNM
Vorbis	MOVEMENTNAME
APEv2	MOVEMENTNAME
iTunes MP4	©mvn
ASF/Windows Media	n/a
RIFF INFO	n/a

19.2.42 Movement Count ^[4]

Internal Name	movementtotal (Picard>=2.1)
ID3v2	MVIN
Vorbis	MOVEMENTTOTAL
APEv2	MOVEMENTTOTAL
iTunes MP4	mvc
ASF/Windows Media	n/a
RIFF INFO	n/a

19.2.43 Movement Number ^[4]

Internal Name	movementnumber (Picard>=2.1)
ID3v2	MVIN
Vorbis	MOVEMENT
APEv2	MOVEMENT
iTunes MP4	mvi
ASF/Windows Media	n/a
RIFF INFO	n/a

19.2.44 MusicBrainz Artist ID

Internal Name	musicbrainz_artistid
ID3v2	TXXX:MusicBrainz Artist Id
Vorbis	MUSICBRAINZ_ARTISTID
APEv2	MUSICBRAINZ_ARTISTID
iTunes MP4	----:com.apple.iTunes:MusicBrainz Artist Id
ASF/Windows Media	MusicBrainz/Artist Id
RIFF INFO	n/a

19.2.45 MusicBrainz Disc ID

Internal Name	musicbrainz_discid
ID3v2	TXXX:MusicBrainz Disc Id
Vorbis	MUSICBRAINZ_DISCID
APEv2	MUSICBRAINZ_DISCID
iTunes MP4	----:com.apple.iTunes:MusicBrainz Disc Id
ASF/Windows Media	MusicBrainz/Disc Id
RIFF INFO	n/a

19.2.46 MusicBrainz Original Artist ID

Internal Name	musicbrainz_originalartistid
ID3v2	TXXX:MusicBrainz Original Artist Id
Vorbis	MUSICBRAINZ_ORIGINALARTISTID
APEv2	n/a
iTunes MP4	----:com.apple.iTunes:MusicBrainz Original Artist Id (Picard>=2.1)
ASF/Windows Media	MusicBrainz/Original Artist Id (Picard>=2.1)
RIFF INFO	n/a

19.2.47 MusicBrainz Original Release ID

Internal Name	musicbrainz_originalalbumid
ID3v2	TXXX:MusicBrainz Original Album Id
Vorbis	MUSICBRAINZ_ORIGINALALBUMID
APEv2	n/a
iTunes MP4	----:com.apple.iTunes:MusicBrainz Original Album Id (Picard>=2.1)
ASF/Windows Media	MusicBrainz/Original Album Id (Picard>=2.1)
RIFF INFO	n/a

19.2.48 MusicBrainz Recording ID

Internal Name	musicbrainz_recordingid
ID3v2	UFID:http://musicbrainz.org
Vorbis	MUSICBRAINZ_TRACKID
APEv2	MUSICBRAINZ_TRACKID
iTunes MP4	----:com.apple.iTunes:MusicBrainz Track Id
ASF/Windows Media	MusicBrainz/Track Id
RIFF INFO	n/a

19.2.49 MusicBrainz Release Artist ID

Internal Name	musicbrainz_albumartistid
ID3v2	TXXX:MusicBrainz Album Artist Id
Vorbis	MUSICBRAINZ_ALBUMARTISTID
APEv2	MUSICBRAINZ_ALBUMARTISTID
iTunes MP4	----:com.apple.iTunes:MusicBrainz Album Artist Id
ASF/Windows Media	MusicBrainz/Album Artist Id
RIFF INFO	n/a

19.2.50 MusicBrainz Release Group ID

Internal Name	musicbrainz_releasegroupid
ID3v2	TXXX:MusicBrainz Release Group Id
Vorbis	MUSICBRAINZ_RELEASEGROUPID
APEv2	MUSICBRAINZ_RELEASEGROUPID
iTunes MP4	----:com.apple.iTunes:MusicBrainz Release Group Id
ASF/Windows Media	MusicBrainz/Release Group Id
RIFF INFO	n/a

19.2.51 MusicBrainz Release ID

Internal Name	musicbrainz_albumid
ID3v2	TXXX:MusicBrainz Album Id
Vorbis	MUSICBRAINZ_ALBUMID
APEv2	MUSICBRAINZ_ALBUMID
iTunes MP4	----:com.apple.iTunes:MusicBrainz Album Id
ASF/Windows Media	MusicBrainz/Album Id
RIFF INFO	n/a

19.2.52 MusicBrainz Track ID

Internal Name	musicbrainz_trackid
ID3v2	TXXX:MusicBrainz Release Track Id
Vorbis	MUSICBRAINZ_RELEASETRACKID
APEv2	MUSICBRAINZ_RELEASETRACKID
iTunes MP4	----:com.apple.iTunes:MusicBrainz Release Track Id
ASF/Windows Media	MusicBrainz/Release Track Id
RIFF INFO	n/a

19.2.53 MusicBrainz TRM ID

Internal Name	musicbrainz_trmid (deprecated)
ID3v2	TXXX:MusicBrainz TRM Id
Vorbis	MUSICBRAINZ_TRMID
APEv2	MUSICBRAINZ_TRMID
iTunes MP4	----:com.apple.iTunes:MusicBrainz TRM Id
ASF/Windows Media	MusicBrainz/TRM Id
RIFF INFO	n/a

19.2.54 MusicBrainz Work ID

Internal Name	musicbrainz_workid
ID3v2	TXXX:MusicBrainz Work Id
Vorbis	MUSICBRAINZ_WORKID
APEv2	MUSICBRAINZ_WORKID
iTunes MP4	----:com.apple.iTunes:MusicBrainz Work Id
ASF/Windows Media	MusicBrainz/Work Id
RIFF INFO	n/a

19.2.55 MusicIP Fingerprint

Internal Name	musicip_fingerprint
ID3v2	TXXX:MusicMagic Fingerprint
Vorbis	FINGERPRINT=MusicMagic Fingerprint {fingerprint}
APEv2	n/a
iTunes MP4	----:com.apple.iTunes:fingerprint
ASF/Windows Media	n/a
RIFF INFO	n/a

19.2.56 MusicIP PUID

Internal Name	musicip_puid
ID3v2	TXXX:MusicIP PUID
Vorbis	MUSICIP_PUID
APEv2	MUSICIP_PUID
iTunes MP4	----:com.apple.iTunes:MusicIP PUID
ASF/Windows Media	MusicIP/PUID
RIFF INFO	n/a

19.2.57 Original Album

Internal Name	originalalbum
ID3v2	TOAL
Vorbis	n/a
APEv2	n/a
iTunes MP4	n/a
ASF/Windows Media	WM/OriginalAlbumTitle (Picard>=2.1)
RIFF INFO	n/a

19.2.58 Original Artist

Internal Name	originalartist
ID3v2	TOPE
Vorbis	n/a
APEv2	Original Artist (Picard>=2.1)
iTunes MP4	n/a
ASF/Windows Media	WM/OriginalArtist (Picard>=2.1)
RIFF INFO	n/a

19.2.59 Original Filename

Internal Name	originalfilename (Picard>=2.3)
ID3v2	TOFN
Vorbis	ORIGINALFILENAME
APEv2	ORIGINALFILENAME
iTunes MP4	n/a
ASF/Windows Media	WM/OriginalFilename
RIFF INFO	n/a

19.2.60 Original Release Date ^[1]

Internal Name	originaldate
ID3v2	TDOR (ID3v2.4) TORY (ID3v2.3)
Vorbis	ORIGINALDATE
APEv2	n/a
iTunes MP4	n/a
ASF/Windows Media	WM/OriginalReleaseTime (Picard>=1.3.1) WM/ OriginalReleaseYear (Picard<=1.3.0)
RIFF INFO	n/a

19.2.61 Original Release Year ^[1]

Internal Name	originalyear
ID3v2	n/a
Vorbis	ORIGINALYEAR
APEv2	ORIGINALYEAR
iTunes MP4	n/a
ASF/Windows Media	WM/OriginalReleaseYear (Picard>=1.3.1)
RIFF INFO	n/a

19.2.62 Performer

Internal Name	performer:instrument
ID3v2	TMCL:instrument (ID3v2.4) IPLS:instrument (ID3v2.3)
Vorbis	PERFORMER={artist} (instrument)
APEv2	Performer={artist} (instrument)
iTunes MP4	n/a
ASF/Windows Media	n/a
RIFF INFO	n/a

See also:

Please refer to [Relationship Types / Artist-Release / Performer](#) , [Relationship Types / Artist-Release / Vocal](#) , [Relationship Types / Artist-Release / Instrument](#) , [Relationship Types / Artist-Recording / Performer](#) , [Relationship Types / Artist-Recording / Vocal](#) , and [Relationship Types / Artist-Recording / Instrument](#) for more information.

19.2.63 Podcast ^[4]

Internal Name	podcast
ID3v2	n/a
Vorbis	n/a
APEv2	n/a
iTunes MP4	pcst
ASF/Windows Media	n/a
RIFF INFO	n/a

19.2.64 Podcast URL ^[4]

Internal Name	podcasturl
ID3v2	n/a
Vorbis	n/a
APEv2	n/a
iTunes MP4	purl
ASF/Windows Media	n/a
RIFF INFO	n/a

19.2.65 Producer

Internal Name	producer
ID3v2	TIPL:producer (ID3v2.4) IPLS:producer (ID3v2.3)
Vorbis	PRODUCER
APEv2	Producer
iTunes MP4	----:com.apple.iTunes:PRODUCER
ASF/Windows Media	WM/Producer
RIFF INFO	IPRO

19.2.66 Rating

Internal Name	_rating
ID3v2	POPM
Vorbis	RATING:user@email
APEv2	n/a
iTunes MP4	n/a
ASF/Windows Media	WM/SharedUserRating
RIFF INFO	n/a

19.2.67 Record Label

Internal Name	label
ID3v2	TPUB
Vorbis	LABEL
APEv2	Label
iTunes MP4	----:com.apple.iTunes:LABEL
ASF/Windows Media	WM/Publisher
RIFF INFO	n/a

19.2.68 Release Country

Internal Name	releasecountry
ID3v2	TXXX:MusicBrainz Album Release Country
Vorbis	RELEASECOUNTRY
APEv2	RELEASECOUNTRY
iTunes MP4	----:com.apple.iTunes:MusicBrainz Album Release Country
ASF/Windows Media	MusicBrainz/Album Release Country
RIFF INFO	ICNT

19.2.69 Release Date ^[10]

Internal Name	releasedate (since Picard 2.9, not filled by default)
ID3v2	TDRL (ID3v2.4) TXXX:RELEASEDATE (ID3v2.3)
Vorbis	RELEASEDATE
APEv2	RELEASEDATE
iTunes MP4	----:com.apple.iTunes:RELEASEDATE
ASF/Windows Media	n/a
RIFF INFO	n/a

19.2.70 Release Status

Internal Name	releasestatus
ID3v2	TXXX:MusicBrainz Album Status
Vorbis	RELEASESTATUS
APEv2	MUSICBRAINZ_ALBUMSTATUS
iTunes MP4	----:com.apple.iTunes:MusicBrainz Album Status
ASF/Windows Media	MusicBrainz/Album Status
RIFF INFO	n/a

19.2.71 Release Type

Internal Name	releasetype
ID3v2	TXXX:MusicBrainz Album Type
Vorbis	RELEASETYPE
APEv2	MUSICBRAINZ_ALBUMTYPE
iTunes MP4	----:com.apple.iTunes:MusicBrainz Album Type
ASF/Windows Media	MusicBrainz/Album Type
RIFF INFO	n/a

19.2.72 Remixer

Internal Name	remixer
ID3v2	TPE4
Vorbis	REMIXER
APEv2	MixArtist
iTunes MP4	----:com.apple.iTunes:REMIXER
ASF/Windows Media	WM/ModifiedBy
RIFF INFO	n/a

19.2.73 ReplayGain Album Gain

Internal Name	replaygain_album_gain (Picard>=2.2)
ID3v2	TXXX:REPLAYGAIN_ALBUM_GAIN
Vorbis	REPLAYGAIN_ALBUM_GAIN
APEv2	REPLAYGAIN_ALBUM_GAIN
iTunes MP4	----:com.apple.iTunes:REPLAYGAIN_ALBUM_GAIN
ASF/Windows Media	REPLAYGAIN_ALBUM_GAIN
RIFF INFO	n/a

19.2.74 ReplayGain Album Peak

Internal Name	replaygain_album_peak (Picard>=2.2)
ID3v2	TXXX:REPLAYGAIN_ALBUM_PEAK
Vorbis	REPLAYGAIN_ALBUM_PEAK
APEv2	REPLAYGAIN_ALBUM_PEAK
iTunes MP4	----:com.apple.iTunes:REPLAYGAIN_ALBUM_PEAK
ASF/Windows Media	REPLAYGAIN_ALBUM_PEAK
RIFF INFO	n/a

19.2.75 ReplayGain Album Range

Internal Name	replaygain_album_range (Picard>=2.2)
ID3v2	TXXX:REPLAYGAIN_ALBUM_RANGE
Vorbis	REPLAYGAIN_ALBUM_RANGE
APEv2	REPLAYGAIN_ALBUM_RANGE
iTunes MP4	----:com.apple.iTunes:REPLAYGAIN_ALBUM_RANGE
ASF/Windows Media	REPLAYGAIN_ALBUM_RANGE
RIFF INFO	n/a

19.2.76 ReplayGain Reference Loudness

Internal Name	replaygain_reference_loudness (Picard>=2.2)
ID3v2	TXXX:REPLAYGAIN_REFERENCE_LOUDNESS
Vorbis	REPLAYGAIN_REFERENCE_LOUDNESS
APEv2	REPLAYGAIN_REFERENCE_LOUDNESS
iTunes MP4	----:com.apple.iTunes:REPLAYGAIN_REFERENCE_LOUDNESS
ASF/Windows Media	REPLAYGAIN_REFERENCE_LOUDNESS
RIFF INFO	n/a

19.2.77 ReplayGain Track Gain

Internal Name	replaygain_track_gain (Picard>=2.2)
ID3v2	TXXX:REPLAYGAIN_TRACK_GAIN
Vorbis	REPLAYGAIN_TRACK_GAIN
APEv2	REPLAYGAIN_TRACK_GAIN
iTunes MP4	----:com.apple.iTunes:REPLAYGAIN_TRACK_GAIN
ASF/Windows Media	REPLAYGAIN_TRACK_GAIN
RIFF INFO	n/a

19.2.78 ReplayGain Track Peak

Internal Name	replaygain_track_peak (Picard>=2.2)
ID3v2	TXXX:REPLAYGAIN_TRACK_PEAK
Vorbis	REPLAYGAIN_TRACK_PEAK
APEv2	REPLAYGAIN_TRACK_PEAK
iTunes MP4	----:com.apple.iTunes:REPLAYGAIN_TRACK_PEAK
ASF/Windows Media	REPLAYGAIN_TRACK_PEAK
RIFF INFO	n/a

19.2.79 ReplayGain Track Range

Internal Name	replaygain_track_range (Picard>=2.2)
ID3v2	TXXX:REPLAYGAIN_TRACK_RANGE
Vorbis	REPLAYGAIN_TRACK_RANGE
APEv2	REPLAYGAIN_TRACK_RANGE
iTunes MP4	----:com.apple.iTunes:REPLAYGAIN_TRACK_RANGE
ASF/Windows Media	REPLAYGAIN_TRACK_RANGE
RIFF INFO	n/a

19.2.80 Script

Internal Name	script
ID3v2	TXXX:SCRIPT
Vorbis	SCRIPT
APEv2	Script
iTunes MP4	----:com.apple.iTunes:SCRIPT
ASF/Windows Media	WM/Script
RIFF INFO	n/a

19.2.81 Show Name ^[4]

Internal Name	show
ID3v2	n/a
Vorbis	n/a
APEv2	n/a
iTunes MP4	tvsh
ASF/Windows Media	n/a
RIFF INFO	n/a

19.2.82 Show Name Sort Order ^[4]

Internal Name	showsort
ID3v2	n/a
Vorbis	n/a
APEv2	n/a
iTunes MP4	sosn
ASF/Windows Media	n/a
RIFF INFO	n/a

19.2.83 Show Work & Movement ^[4]

Internal Name	showmovement (Picard>=2.1)
ID3v2	TXXX:SHOWMOVEMENT
Vorbis	SHOWMOVEMENT
APEv2	SHOWMOVEMENT
iTunes MP4	shwm
ASF/Windows Media	n/a
RIFF INFO	n/a

19.2.84 Subtitle ^[4]

Internal Name	subtitle
ID3v2	TIT3
Vorbis	SUBTITLE
APEv2	Subtitle
iTunes MP4	----:com.apple.iTunes:SUBTITLE
ASF/Windows Media	WM/SubTitle
RIFF INFO	n/a

19.2.85 Total Discs

Internal Name	totaldiscs
ID3v2	TP05
Vorbis	DISCTOTAL and TOTALDISCS
APEv2	Disc
iTunes MP4	disk
ASF/Windows Media	WM/PartOfSet (Picard>=1.3.1)
RIFF INFO	n/a

19.2.86 Total Tracks

Internal Name	totaltracks
ID3v2	TRCK
Vorbis	TRACKTOTAL and TOTALTRACKS
APEv2	Track
iTunes MP4	trkn
ASF/Windows Media	n/a
RIFF INFO	n/a

19.2.87 Track Number

Internal Name	tracknumber
ID3v2	TRCK
Vorbis	TRACKNUMBER
APEv2	Track
iTunes MP4	trkn
ASF/Windows Media	WM/TrackNumber
RIFF INFO	ITRK

19.2.88 Track Title

Internal Name	title
ID3v2	TIT2
Vorbis	TITLE
APEv2	Title
iTunes MP4	©nam
ASF/Windows Media	Title
RIFF INFO	INAM

19.2.89 Track Title Sort Order ^[4]

Internal Name	titlesort
ID3v2	TSOT
Vorbis	TITLESORT
APEv2	TITLESORT
iTunes MP4	sonm
ASF/Windows Media	WM/TitleSortOrder
RIFF INFO	n/a

19.2.90 Website (official artist website)

Internal Name	website
ID3v2	WOAR
Vorbis	WEBSITE
APEv2	Weblink
iTunes MP4	n/a
ASF/Windows Media	WM/AuthorURL (Picard>=1.3.1)
RIFF INFO	n/a

19.2.91 Work Title

Internal Name	work (Picard>=1.3)
ID3v2	TXXX:WORK TIT1 ^[8]
Vorbis	WORK
APEv2	WORK
iTunes MP4	©wrk (Picard>=2.1)
ASF/Windows Media	WM/Work
RIFF INFO	n/a

19.2.92 Writer ^[2]

Internal Name	writer
ID3v2	TXXX:Writer (Picard>=1.3)
Vorbis	WRITER
APEv2	Writer
iTunes MP4	n/a
ASF/Windows Media	n/a
RIFF INFO	IWRI

Notes:

1. Taken from the earliest release in the release group.
2. Used when uncertain whether composer or lyricist.
3. This is populated by LastFMPlus plugin and not by stock Picard.
4. This is not able to be populated by stock Picard. It may be used and populated by certain plugins.
5. For Picard>=1.3 this indicates a Various Artists album; for Picard<=1.2 this indicates albums with tracks by different artists which is incorrect (e.g.: an original album with a duet with a feat. artist would show as a Compilation). In neither case does this indicate a MusicBrainz Release Group subtype of compilation.
6. [Release-level license](#) relationship type.
7. [Recording-level license](#) relationship type.
8. With “Save iTunes compatible grouping and work” (since Picard>=2.1.0)
9. From iTunes Metadata Format Specification
10. For compatibility reasons the date tag gets filled with the release date from MusicBrainz. This is how most software interprets this tag. Since Picard 2.9 the separate releasedate exists for use by scripts and plugins, but is not filled by default.

19.3 Appendix C: Command Line Options

Picard can be started from the command line with the following arguments:

```
run_picard.py [-h] [-a AUDIT] [-c CONFIG_FILE] [-d] [-e COMMAND]
→[COMMAND ...] [-M] [-N] [-P] [--no-crash-dialog] [-s] [-v] [-V]
→[FILE_OR_URL ...]
```

where the options are:

- h, --help**
show a help message and exit
 - a AUDIT, --audit AUDIT**
audit events passed as a comma-separated list, prefixes supported, use “all” to match any event. See the [Python Documentation](#) for more information.
 - c CONFIG_FILE, --config-file CONFIG_FILE**
location of the configuration file to use
 - d, --debug**
enable debug-level logging
 - e COMMAND, --exec COMMAND**
execute one or more COMMANDs at start-up (see [Executable Commands](#) for more information)
 - M, --no-player**
disable built-in media player
 - N, --no-restore**
do not restore window positions or sizes
 - P, --no-plugins**
do not load any plugins
 - no-crash-dialog**
disable the crash dialog
 - s, --stand-alone-instance**
force Picard to create a new, stand-alone instance
 - v, --version**
display the version information and exit
 - V, --long-version**
display the long version information and exit
- FILE_OR_URL**
one or more files, directories, URLs and MBIDs to load

Note: Files and directories are specified including the path (either absolute or relative) to the file or directory, and may include drive specifiers. They can also be specified using the `file://` prefix. URLs are specified by using either the `http://` or `https://` prefix. MBIDs are specified in the format `mbid://<entity_type>/<mbid>` where `<entity_type>` is one of “release”, “artist” or “track” and `<mbid>` is the MusicBrainz Identifier of the entity.

If a specified item contains a space, it must be enclosed in quotes such as `"/home/user/music/my song.mp3"`.

19.4 Appendix D: Keyboard Shortcuts

In addition to the standard keyboard shortcuts provided by your operating system for things like text selection, copy and paste, Picard also provides the following:

19.4.1 Main window

File

Action	Windows / Linux	macOS
Add folder	Ctrl+E	⌘+E
Add files	Ctrl+O	⌘+O
Save selected files	Ctrl+S	⌘+S
Quit Picard	Ctrl+Q	⌘+Q

Edit

Action	Windows / Linux	macOS
Cut selected files	Ctrl+X	⌘+X
Paste selected files	Ctrl+V	⌘+V
Show info for selected item	Ctrl+I	⌘+I

View

Action	Windows / Linux	macOS
Toggle file browser	Ctrl+B	⌘+B
Toggle metadata view	Ctrl+Shift+M	⌘+⇧+M

Options

Action	Windows / Linux	macOS
Open file naming script editor	Ctrl+Shift+S	⌘+⇧+S
Open profile editor	Ctrl+Shift+P	⌘+⇧+P

Tools

Action	Windows / Linux	macOS
Refresh	Ctrl+R	⌘+R
Lookup CD	Ctrl+K	⌘+K
Lookup	Ctrl+L	⌘+L
Scan	Ctrl+Y	⌘+Y
Cluster	Ctrl+U	⌘+U
Lookup in browser	Ctrl+Shift+L	⌘+⇧+L
Search for similar albums / tracks	Ctrl+T	⌘+T
Show other album versions	Ctrl+Shift+0	⌘+⇧+0
Generate AcoustID fingerprints	Ctrl+Shift+Y	⌘+⇧+Y
Tags from file names	Ctrl+Shift+T	⌘+⇧+T

Help

Action	Windows / Linux	macOS
Help	F1	⌘+?
View activity history	Ctrl+H	⌘+⇧+H
View error/debug log	Ctrl+G	⌘+G

Metadata view

Action	Windows / Linux	macOS
Add new tag	Alt+Shift+A	⌘+⇧+A
Edit selected tag	Alt+Shift+E	⌘+⇧+E
Remove selected tag	Alt+Shift+R Del	⌘+⇧+R Del ⌘+⌫
Copy selected tag value	Ctrl+C	⌘+C
Paste to selected tag value	Ctrl+V	⌘+V

Other

Action	Windows / Linux	macOS
Focus search	Ctrl+F	⌘+F
Remove selected item	Del	Del ⌘+⌫

19.4.2 Script editor

Action	Windows / Linux	macOS
Show auto completion	Ctrl+Space	^+Space
Use selected completion	Tab Return	Tab Return
Hide completions	Esc	Esc

19.4.3 File naming script editor

Action	Windows / Linux	macOS
Show auto completion	Ctrl+Space	^+Space
Use selected completion	Tab Return	Tab Return
Hide completions	Esc	Esc
Edit script metadata	Ctrl+M	⌘+M
Word wrap on/off	Ctrl+Shift+W	⌘+⇧+W
Show/hide help tooltips	Ctrl+Shift+T	⌘+⇧+T
Show/hide documentation	Ctrl+H	⌘+H
Help (in browser)	F1	⌘+?

Symbols

- M
 - command line option, 294
- N
 - command line option, 294
- P
 - command line option, 294
- V
 - command line option, 294
- audit
 - command line option, 294
- config-file
 - command line option, 294
- debug
 - command line option, 294
- exec
 - command line option, 294
- help
 - command line option, 294
- long-version
 - command line option, 294
- no-crash-dialog
 - command line option, 294
- no-player
 - command line option, 294
- no-plugins
 - command line option, 294
- no-restore
 - command line option, 294
- stand-alone-instance
 - command line option, 294
- version
 - command line option, 294
- a
 - command line option, 294
- c
 - command line option, 294

- d
 - command line option, 294
- e
 - command line option, 294
- h
 - command line option, 294
- s
 - command line option, 294
- v
 - command line option, 294

A

- acknowledgements, 4
- acoustic
 - fingerprint, 241
- acoustic fingerprint, 6, 236
 - submitting, 179, 181, 182, 188
- AcoustID, 6
 - automatic scan, 25
 - submitting, 188
 - usage, 241
- album
 - load error, 230
- albumartist, 6
- android
 - app, 250
- api
 - plugins, 263
- artist, 6
- artist credit, 7
- audio player, 235
- automatic clustering
 - cluster, 25
- automatic scan
 - AcoustID, 25

B

- batch processing, 207

- browser
 - configuration, [237](#)
- C
- CAA, see cover art archive
- cluster
 - automatic clustering, [25](#)
 - lookup, [162](#)
 - submitting, [196](#)
- command line
 - options, [294](#)
- command line option
 - M, [294](#)
 - N, [294](#)
 - P, [294](#)
 - V, [294](#)
 - audit, [294](#)
 - config-file, [294](#)
 - debug, [294](#)
 - exec, [294](#)
 - help, [294](#)
 - long-version, [294](#)
 - no-crash-dialog, [294](#)
 - no-player, [294](#)
 - no-plugins, [294](#)
 - no-restore, [294](#)
 - stand-alone-instance, [294](#)
 - version, [294](#)
 - a, [294](#)
 - c, [294](#)
 - d, [294](#)
 - e, [294](#)
 - h, [294](#)
 - s, [294](#)
 - v, [294](#)
- command processing, [207](#)
- commands
 - executable, [209](#)
- configuration
 - aac tag options, [40](#)
 - ac3 tag options, [41](#)
 - action options, [23](#)
 - advanced options, [74](#)
 - before tagging, [37](#)
 - browser, [237](#)
 - cd lookup, [58](#)
 - colors, [67](#)
 - config file location, [236](#)
 - cover art, [43](#)
 - cover art archive, [46](#)
 - cover art location, [44](#)
 - cover art providers, [45](#)
 - file, [236](#)
 - file naming, [49](#)
 - file naming compatibility, [55](#)
 - fingerprinting, [57](#)
 - general options, [24](#)
 - genres, [34](#)
 - id3 tag options, [38](#)
 - local files, [47](#)
 - maintenance, [79](#)
 - matching preferences, [77](#)
 - metadata options, [29](#)
 - network, [76](#)
 - plugin update checking, [26](#)
 - plugins, [60](#)
 - profiles, [27](#)
 - program update checking, [26](#)
 - ratings, [36](#)
 - release preferences, [32](#)
 - screen setup, [22](#)
 - scripting, [72](#)
 - tag options, [37](#)
 - toolbar, [70](#)
 - top tags, [68](#)
 - user interface, [64](#)
 - wave tag options, [42](#)
- context menu actions
 - plugins, [268](#)
- contributing, [3](#)
- cover art
 - configuration, [43](#)
 - location to save, [44](#)
 - setting, [174](#)
- cover art archive, [7](#)
 - configuration, [46](#)
- D
- dBpoweramp
 - lookup log, [180](#)
- disc id, [7](#)
 - attaching, [185](#)
- E
- EAC
 - lookup log, [180](#)

- error
 - couldn't load album, 230
- event hooks
 - plugins, 265
- executable commands, 209
 - CLEAR_LOGS, 209
 - CLUSTER, 209
 - FINGERPRINT, 209
 - FROM_FILE, 209
 - LOAD, 211
 - LOOKUP, 211
 - LOOKUP_CD, 211
 - PAUSE, 212
 - QUIT, 212
 - REMOVE, 212
 - REMOVE_ALL, 212
 - REMOVE_EMPTY, 213
 - REMOVE_SAVED, 213
 - REMOVE_UNCLUSTERED, 213
 - SAVE_MATCHED, 213
 - SAVE_MODIFIED, 213
 - SCAN, 214
 - SHOW, 214
 - SUBMIT_FINGERPRINTS, 214
 - WRITE_LOGS, 214
- F
- file format
 - plugins, 267
- file formats, 232
- file naming
 - configuration, 49
 - script editor, 52
 - scripts, 216
- file naming compatibility
 - configuration, 55
- file naming script, 238
- files
 - saving, 176
- fingerprint
 - acoustic, 6, 236, 241
 - submitting, 188
- flatpak
 - install, 10
- fre:ac
 - lookup log, 180

- G
- glossary, 6
- I
- icon
 - tagger, 230
- icons
 - album, 19
 - release, 19
 - status, 19
 - status bar, 21
 - track, 20
- identifier
 - musicbrainz, 81, 87, 90, 93
- install
 - download, 10
 - flatpak, 10
 - Linux package, 11
 - snap, 11
- itunes, 236
 - tags, 236
- L
- limitations, 2
- Linux package
 - install, 11
- lookup
 - cd, 158
 - cluster, 162
 - ripper log, 158, 180
- lookup files, 161
- lookup in browser, 166
- M
- macos
 - app damaged, 231
 - network folders, 231
- mapping
 - tags, 270
- matching files to tracks, 172
- mbid, see MusicBrainz Identifier, 81, 87, 90, 93
- medium, 7
- multiple release countries, 243
- MusicBrainz Identifier, 7

N

non-album track, see standalone recording

O

option settings, see configuration

options

command line, 294

P

plugin

writing, 245

plugin update checking

configuration, 26

plugins, 215

api, 263

configuration, 60

context menu actions, 268

event hooks, 265

file format, 267

installing, 63

metadata, 263

metadata processors, 264

programming, 263

scripting functions, 267

third party, 61

types, 215

processing

batch, 207

processing order, 220

profiles

option, 27

program update checking

configuration, 26

programming

plugins, 263

R

rate limiting, 2

recording, 8

standalone, 8

release, 8

deleted, 231

load error, 230

submitting, 196

release countries

multiple, 243

release group, 8

S

saving files, 176

scan files, 164

script

file naming, 238

script editor

file naming, 52

scripting

functions, 103

scripting functions

assignment, 103

conditional, 132

information, 146

loop, 155

mathematical, 129

miscellaneous, 156

multi-value, 120

plugins, 267

text, 107

scripts, 101, 216

file naming, 52, 216

syntax, 101

tagging, 72, 217

tags, 81

variables, 81

snap

install, 11

standalone recording, 8

starting, 12

status bar, 21

T

tagger icon, 230

tagging

itunes, 236

scripts, 217

tags

advanced, 87

basic, 81

classical, 96

editing, 234

genre, 90

mapping, 270

plugins, 96

truncated, 237

writing, 100

tags from file names, 193

track, 8

troubleshooting

- files not saved, [228](#)
- general, [222](#)
- get debug log, [223](#)
- getting log for crashes, [223](#)
- macOS shows app is damaged, [229](#)
- no cover art displayed, [227](#)
- no cover art downloaded, [226](#)
- program freezes, [228](#)
- program won't start, [225](#)
- reporting a bug, [222](#)
- tags not saved, [227](#)

U**user interface**

- colors, [67](#)
- configuration, [64](#)
- keyboard shortcuts, [296](#)
- main screen, [13](#)
- toolbar, [70](#)

V**variables**

- advanced, [93](#)
- basic, [90](#)
- file, [92](#)

W**Whipper**

- lookup log, [180](#)

WMP

- tags, [237](#)

work, [9](#)**workflows**

- cd, [178](#)
- files grouped by album, [181](#)
- files not grouped, [182](#)
- general, [178](#)
- no metadata, [183](#)
- ripper log, [180](#)

X**XLD**

- lookup log, [180](#)