
MusicBrainz Picard

Release v2.3.1

Apr 26, 2020

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.

My sincere thanks to all the volunteers for your time and effort that have helped make this documentation project a reality.

Bob Swift (rdswift)
editor

Copyright © 2020 Bob Swift (rdswift).
Copyright © 2020 MetaBrainz Foundation.

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

CONTENTS

1	Introduction	1
1.1	Picard Can.	1
1.2	Picard Cannot.	2
1.3	Limitations	2
2	Glossary of Terms	3
3	Getting Started	6
3.1	Download & Install	6
3.2	Main Screen	7
4	Configuration	8
4.1	Screen Setup	8
4.2	Action Options	9
4.3	Option Settings	9
5	Tags & Variables	25
6	Scripting	26
6.1	Syntax	26
6.2	Metadata Variables	26
6.3	Scripting Functions	26
7	Scripting Functions	27
7.1	Assignment Functions	27
7.2	Text Functions	30
7.3	Multi-Value Functions	41
7.4	Mathematical Functions	47
7.5	Conditional Functions	50
7.6	Information Functions	61
7.7	Loop Functions	62
7.8	Miscellaneous Functions	63
8	Using Picard	64
9	Extending Picard	65

9.1	Plugins	65
9.2	Scripts	66
9.3	Processing Order	67
10	Examples	70
11	Appendices	71
11.1	Plugins API	71

INTRODUCTION

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

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

1.1 Picard Can...

... add metadata tags to your music files, based on information available from the [MusicBrainz website](https://musicbrainz.org/)².

... 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](https://acoustid.org/)³.

¹ <https://picard.musicbrainz.org/>

² <https://musicbrainz.org/>

³ <https://acoustid.org/>

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 two notable exceptions: Matroska (.mka) and Windows WAVE (.wav). WAVs cannot be tagged due to the lack of a standard for doing so, however, they can be fingerprinted and renamed. In addition, Picard does not support writing custom tags for all formats. The [Picard Tag Mappings](https://picard.musicbrainz.org/docs/mappings/)⁴ webpage provides more information regarding the mapping between Picard internal tag names and various tagging formats.

Request Rate Limiting

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

Network File Processing

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

⁴ <https://picard.musicbrainz.org/docs/mappings/>

GLOSSARY OF TERMS

Many of the terms used in this documentation and within Picard itself have specific meaning in the 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](https://wikipedia.org/wiki/Acoustic_fingerprint)⁵ for a full explanation of acoustic fingerprinting.

AcoustID

AcoustID is an acoustic fingerprint system built entirely on open-source technology. See the [AcoustID website](https://acoustid.org/)⁶ for additional information.

albumartist

The musician or group of musicians performing on a release. Note that this is different for Classical Music releases, which follow the MusicBrainz [Classical Style Guide](https://musicbrainz.org/doc/Style/Classical)⁷, listing the composer(s) first, followed by the performers.

artist

The musician or group of musicians performing on a track. Note that this is different for Classical Music releases, which follow the MusicBrainz Classical Style Guide, showing only the composer and not the performers. Please see the [Artist](https://musicbrainz.org/doc/Artist)⁸ page on the MusicBrainz website for additional information.

artist credit

Artist credits indicate who is the main credited artist (or artists) for releases, release groups, tracks and recordings, and how they are credited. They consist of artists, with (optionally) their names as credited in the specific release, track, etc., and join phrases between them. Please see the [Artist Credits](https://musicbrainz.org/doc/Artist_Credits)⁹ page on the MusicBrainz website for additional information.

caa

⁵ https://wikipedia.org/wiki/Acoustic_fingerprint

⁶ <https://acoustid.org/>

⁷ <https://musicbrainz.org/doc/Style/Classical>

⁸ <https://musicbrainz.org/doc/Artist>

⁹ https://musicbrainz.org/doc/Artist_Credits

The [Cover Art Archive](https://coverartarchive.org/)¹⁰ which is a joint project between the [Internet Archive](https://archive.org/)¹¹ and [MusicBrainz](https://musicbrainz.org/)¹², whose goal is to make cover art images available to everyone on the Internet in an organised and convenient way. Please see the [Cover Art Archive page](https://musicbrainz.org/doc/Cover_Art_Archive)¹³ on the MusicBrainz website for additional information.

mbid

The MusicBrainz Identifier, which is a unique code used to identify each element in the MusicBrainz database.

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. Please see the [Recording](https://musicbrainz.org/doc/Recording)¹⁴ 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. Please see the [Release](https://musicbrainz.org/doc/Release)¹⁵ 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. Please see the [Release Group](https://musicbrainz.org/doc/Release_Group)¹⁶ page on the MusicBrainz website for additional information.

standalone recording

A recording that is not linked to any release. Please see the [Standalone Recording](https://musicbrainz.org/doc/Standalone_Recording)¹⁷ 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. Please see the [Track](https://musicbrainz.org/doc/Track)¹⁸ page on the MusicBrainz website for additional information.

work

¹⁰ <https://coverartarchive.org/>

¹¹ <https://archive.org/>

¹² <https://musicbrainz.org/>

¹³ https://musicbrainz.org/doc/Cover_Art_Archive

¹⁴ <https://musicbrainz.org/doc/Recording>

¹⁵ <https://musicbrainz.org/doc/Release>

¹⁶ https://musicbrainz.org/doc/Release_Group

¹⁷ https://musicbrainz.org/doc/Standalone_Recording

¹⁸ <https://musicbrainz.org/doc/Track>

A distinct intellectual or artistic creation, which can be expressed in the form of one or more audio recordings. While a ‘Work’ in MusicBrainz is usually musical in nature, it is not necessarily so. For example, a work could be a novel, play, poem or essay, later recorded as an oratory or audiobook. Please see the [Work](https://musicbrainz.org/doc/Work)¹⁹ page on the MusicBrainz website for additional information.

For more information on these and other terms used, please refer to the [Terminology](https://musicbrainz.org/doc/Terminology)²⁰ page on the MusicBrainz website.

¹⁹ <https://musicbrainz.org/doc/Work>

²⁰ <https://musicbrainz.org/doc/Terminology>

GETTING STARTED

3.1 Download & Install

The latest version of MusicBrainz Picard is always available for download from the [Picard Website](https://picard.musicbrainz.org/downloads/)²¹. This includes installers for all major operating systems (e.g.: Linux, macOS and Windows) as well as the source code.

The Picard website also provides additional information regarding [installing Picard using flatpak](https://picard.musicbrainz.org/docs/linux/)²² on Linux systems.

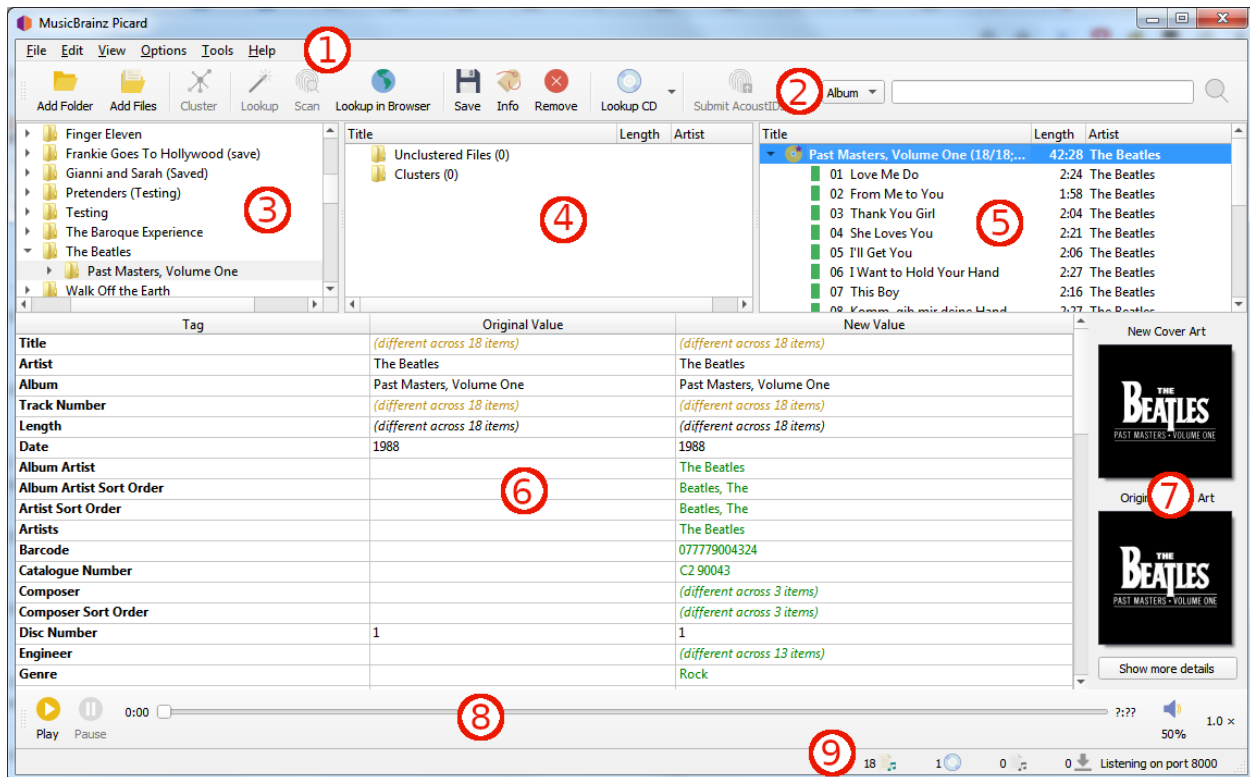
The latest source code is available at the [project repository](https://github.com/musicbrainz/picard)²³ on GitHub.

²¹ <https://picard.musicbrainz.org/downloads/>

²² <https://picard.musicbrainz.org/docs/linux/>

²³ <https://github.com/musicbrainz/picard>

3.2 Main Screen



- 1. Menu Bar:** This provides the pull-down menu of actions that Picard can perform.
- 2. Tool Bar:** This provides quick links to the main functions performed by Picard. This can be customized by the user in the *User Interface Options* settings.
- 3. File Browser:** This provides a browser for selecting files and directories for processing.
- 4. Cluster Pane:** Often referred to as the “left-hand pane”, this section allows the user to select and cluster files for scanning, lookup or matching.
- 5. Album Pane:** Often referred to as the “right-hand pane”, this section displays the albums retrieved from MusicBrainz. This is the section where files are matched to downloaded track information.
- 6. Metadata Pane:** This section is a three-column table of the tag metadata for the album or track currently selected in the Album Pane. The first column shows the tag name, the second shows the original value found in the file, and the third column shows the new value that will be written.
- 7. Cover Art:** This shows the new cover art image that will be written to the selected album or track, along with the original cover art image found in the files matched to the selected album or track.
- 8. Player:** The built-in player that can be used to play selected audio files.
- 9. Status Bar:** The bar at the bottom of the screen shows information about the current operation of Picard, including such items as number of files, albums, and pending downloads.

CONFIGURATION

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

4.1 Screen Setup

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

File Browser

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

Cover Art

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

Actions

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

Search

This displays the manual search box to the right of the “Actions” button bar.

Player

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

4.2 Action Options

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

Rename Files

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

Move Files

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

Save Tags

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

4.3 Option Settings

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

4.3.1 General Options

Server address

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

Port

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

Username

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

Password

Your MusicBrainz website password.

Automatically scan all new files

Check this box if you want Picard to scan each music file you add and look for an AcoustID fingerprint. This takes time, but may be helpful for you and MusicBrainz. Leave it unchecked if you don’t want Picard to do this scan automatically. In any case, you can direct Picard to scan a particular music file at any time using “*Tools* → *Scan*”.

Ignore MBIDs when loading new files

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

Check for updates during start-up

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

Days between checks

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

Updates to check

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

- Stable releases only
- Stable and Beta releases
- Stable, Beta and Dev releases

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

Note: The update settings and “Check for update” function may not be available when Picard is distributed as a package. In that case, the user should check with the maintainer of the package to determine when an update is available.

4.3.2 Metadata Options

Translate artist names to this locale where possible

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

Use standardized artist names

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

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

Use standardized instrument and vocal credits

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

Convert Unicode punctuation characters to ASCII

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

Use release relationships

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

Use track relationships

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

Various artists

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

Non-album tracks

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

Preferred Releases

Preferred release types

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

Preferred release countries

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

Preferred release formats

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

Genres

Use genres from MusicBrainz

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

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

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

Only use my genres

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

Use folksonomy tags as genres

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

Minimal genre usage

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

Maximum number of genres

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

Join multiple genres with

Select which character should be used to separate multiple genres.

Genres or folksonomy tags to include or exclude

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

- **Comments:** Lines not starting with ‘-’ or ‘+’ are ignored. (e.g.: `#comment`, `!comment` or `comment`)
- **Strict filtering:** Exclude exact word by prefixing it with ‘-’ (e.g.: `-word`). Include exact word, even if another rule would exclude it, by prefixing it with ‘+’ (e.g.: `+word`).
- **Wildcard filtering:** Exclude all genres ending with “word” (e.g.: `-*word`). Include all genres starting with “word” (e.g.: `+word*`). Exclude all genres starting with ‘w’ and ending with “rd” (e.g.: `-w*rd`).
- **Regular expressions filtering (Python “re” syntax):** Exclude genres starting with ‘w’ followed by any character, then ‘r’ followed by at least one ‘d’ (e.g.: `-/^w.rd+/`).

Ratings

Enable track ratings

Check to write track ratings to your files.

E-mail

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

Submit ratings to MusicBrainz

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

4.3.3 Tag Options

Write tags to files

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

Preserve timestamps of tagged files

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

Before Tagging

Clear existing tags

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

Remove ID3 tags from FLAC files

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

Remove APEv2 tags from MP3 files

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

Preserve these tags from being cleared or overwritten with MusicBrainz data

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

Tag Compatibility

ID3v2 version

Although ID3v2.4 is the latest version, its support in music players is still lacking. While software such as [foobar2000](https://www.foobar2000.org/)²⁴ and [MediaMonkey](https://www.mediamonkey.com/)²⁵ have no problem using version 2.4 tags, you will not be able to read the tags in Windows Explorer or Windows Media Player (in any Windows or WMP version). Apple iTunes is also still based in ID3v2.3, and support for ID3v2.4 in other media players (such as smartphones) is variable. Other than native support for multi-valued tags in v2.4, the [Picard Tag Mapping](https://picard.musicbrainz.org/docs/mappings/)²⁶ will show you what you lose when choosing v2.3 instead of v2.4.

ID3v2 text encoding

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

Join id3v23 tags with

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

Save iTunes compatible grouping and work

Save the tags grouping and work so that they are compatible with current iTunes versions. Without this option grouping will be displayed in iTunes as “work name” and work will not be available. See the [Picard Tag Mapping page](https://picard.musicbrainz.org/docs/mappings/)²⁷ for details.

Note: For other players supporting grouping and work you might need to disable this option. [MusicBee](https://getmusicbee.com/)²⁸ is one example of this.

Also include ID3v1 tags in the files

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

AAC / AC3 files

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

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

²⁴ <https://www.foobar2000.org/>

²⁵ <https://www.mediamonkey.com/>

²⁶ <https://picard.musicbrainz.org/docs/mappings/>

²⁷ <https://picard.musicbrainz.org/docs/mappings/>

²⁸ <https://getmusicbee.com/>

- **Do not save tags:** Picard will not save any tags to the files, but you can still use Picard to rename them. By default existing APEv2 tags will be kept in the file.
- **Remove APEv2 tags:** If you have “Do not save tags” enabled checking this option will cause Picard to remove existing APEv2 tags from the file on saving.

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

4.3.4 Cover Art Options

Note that you must enable “*Options → Metadata → Use release relationships*” for Picard to be able to download cover art from MusicBrainz cover art relationships.

Location

Embed cover images into tags

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

Only embed a front image

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

Save cover images as separate files

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

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

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

Overwrite the file if it already exists

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

Cover Art Providers

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

Cover Art Archive

The Cover Art Archive (CAA) is the MusicBrainz archive of cover art in cooperation with the [Internet Archive](https://archive.org)²⁹. The Cover Art Archive is the most comprehensive database of cover art (e.g.: front covers, back covers, booklets, CDs).

CAA Release Group

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

Sites on the whitelist

This will use images provided from whitelisted sites. See [Cover art whitelist](#)³⁰ in the Style Guide for more information.

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

Local Files

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](#)³¹.

- **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.tv³², 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](https://www.theaudiodb.com/)³³, which focuses on cover art with high visual quality. This provider provides cover art representative for the release group and not the individual release.

²⁹ <https://archive.org>

³⁰ https://wiki.musicbrainz.org/History:Style/Relationships/URLs/Cover_art_whitelist

³¹ <https://picard.musicbrainz.org/plugins/>

³² <https://fanart.tv/>

³³ <https://www.theaudiodb.com/>

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 Types

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

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

Image Size

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

Save only one front image

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

Download only approved images

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

Use the first image type as the filename

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

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

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

Local Files

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

4.3.5 File Naming Options

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

Move files when saving

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

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

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

Destination directory

This specifies the destination parent 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 are separated by spaces. When these additional files are moved they will end up in the release directory with your music files. In a pattern, the ‘*’ character matches zero or more characters. Other text, like “.jpg”, matches those exact characters. Thus “*.jpg” matches “cover.jpg”, “liner.jpg”, “a.jpg”, and “.jpg”, but not “nomatch.jpg2”.

Delete empty directories

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

Rename files when saving

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

Replace non-ASCII characters

Select this option to replace non-ASCII characters with their ASCII equivalent (e.g.: ‘á’, ‘ä’ and ‘ă’ with ‘a’; ‘é’, ‘ě’ and ‘ë’ with ‘e’; ‘æ’ with “ae”). More information regarding ASCII characters can be found on [Wikipedia](#)³⁴.

Windows compatibility

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

Name files like this

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

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

Scripts are often discussed in the [MetaBrainz Community Forum](#)³⁵, and there is a thread specific to [file naming and script snippets](#)³⁶.

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

4.3.6 Fingerprinting Options

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

MusicBrainz currently supports only [AcoustID](#)³⁷ (an Open Source [acoustic fingerprinting](#)³⁸ system created by [Lukáš Lalinsky](#)³⁹) but has previously supported TRM and MusicID PUID.

Audio Fingerprinting

³⁴ <https://en.wikipedia.org/wiki/ASCII>

³⁵ <https://community.metabrainz.org/>

³⁶ <https://community.metabrainz.org/t/repository-for-neat-file-name-string-patterns-and-tagger-script-snippets/2786/>

³⁷ <https://musicbrainz.org/doc/AcoustID>

³⁸ <https://musicbrainz.org/doc/Fingerprinting>

³⁹ <https://oxygen.sk/>

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.

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](https://acoustid.org/chromaprint)⁴⁰ (fpcalc) utility which is included with the Picard installation.

API key

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

4.3.7 CD Lookup Options

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

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.

macOS

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

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

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

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

Linux

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

Other platforms

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

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.

⁴⁰ <https://acoustid.org/chromaprint>

4.3.8 Plugins Options

This section allows you to manage the plugins used by Picard. You can install new plugins or enable, disable or uninstall plugins that are currently installed. Picard provides a list of plugins that have been submitted to the project. A list of the standard plugins is available on the [plugins page](https://picard.musicbrainz.org/plugins/)⁴¹ 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](https://community.metabrainz.org/)⁴² so if you're looking for a particular enhancement or functionality, a search there might be useful. In addition, one of the MusicBrainz editors, [Colby Dray](https://wiki.musicbrainz.org/User:Colbydray)⁴³ maintains an unofficial list of available plugins on a [wiki page](https://wiki.musicbrainz.org/User:Colbydray/PicardPlugins)⁴⁴.

Note that some plugins have their own option page which will appear under the “Plugins” section of the Options.

4.3.9 User Interface Options

Show text labels under icon

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

Allow selection of multiple directories

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

Use advanced query syntax

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

Show a quit confirmation dialog for unsaved changes

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

Begin browsing in the following directory

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

User interface language

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

Customize action toolbar

⁴¹ <https://picard.musicbrainz.org/plugins/>

⁴² <https://community.metabrainz.org/>

⁴³ <https://wiki.musicbrainz.org/User:Colbydray>

⁴⁴ <https://wiki.musicbrainz.org/User:Colbydray/PicardPlugins>

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

Colors

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

- **Errored entity**: files and other elements with errors on loading or saving
- **Pending entity**: files and other elements queued up for processing
- **Saved entity**: successfully saved files
- **Log view text (debug)**: debug messages in the Error/Debug Log
- **Log view text (error)**: error messages in the Error/Debug Log
- **Log view text (info)**: informational messages in the Error/Debug Log
- **Log view text (warning)**: warning messages in the Error/Debug Log
- **Tag added**: newly added tags in the metadata pane
- **Tag changed**: changed tags in the metadata pane
- **Tag removed**: removed tags in the metadata pane

Top Tags

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

4.3.10 Scripting Options

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

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

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

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

4.3.11 Advanced Options

Ignore file paths matching the following regular expression

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

Ignore hidden files

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

Include sub-folders when adding files from folders

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

Ignore track duration difference under this number of seconds

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

Ignore the following tracks when determining whether a release is complete

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

Tags to ignore for comparison


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

Network

Web Proxy

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

Browser Integration

The browser integration allows you to load releases and recordings into Picard directly from the MusicBrainz website. Once you have opened musicbrainz.org in your browser from Picard, the website will show the green tagger button  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](#)⁴⁵ Android app.

Warning: *Only enable this option 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

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

Minimal similarity for file lookups

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

Minimal similarity for cluster lookups

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

Minimal similarity for matching files to tracks

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

If you have absolutely no metadata in your current files, and you are using “Scan” to match tracks, you may find that you need to lower the value of “Minimal similarity for matching files to tracks” in order for Picard to match the files within a release. Otherwise you may find that Picard matches the track to a release but then is not sure which track is correct; and leaves it in an “unmatched files” group within that release.

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

4.3.12 About

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

⁴⁵ <https://play.google.com/store/apps/details?id=org.musicbrainz.picard.barcodescanner>

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](https://picard.musicbrainz.org/downloads/)⁴⁶ on the website.

⁴⁶ <https://picard.musicbrainz.org/downloads/>

6.1 Syntax

The syntax is derived from Foobar2000's `titleformat`⁴⁷. There are three base elements: text, variable and function. Variables consist of alphanumeric characters enclosed in percent signs (e.g.: `%artist%`). Functions start with a dollar sign and end with an argument list enclosed in parentheses (e.g.: `$lower(...)`).

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

6.2 Metadata Variables

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

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

6.3 Scripting Functions

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

⁴⁷ https://wiki.hydrogenaud.io/index.php?title=Foobar2000:Titleformat_Reference

SCRIPTING FUNCTIONS

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

7.1 Assignment Functions

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

7.1.1 \$copy

Usage: **\$copy(target,source)**

Category: assignment

Implemented: Picard 0.9

Description:

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

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

Example:

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

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

7.1.2 \$copymerge

Usage: **\$copymerge(target,source)**

Category: assignment

Implemented: Picard 1.0

Description:

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

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

Example:

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

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

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

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

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

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

7.1.3 \$delete

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

Category: assignment

Implemented: Picard 2.1

Description:

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

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

Example:

The following statements will perform the actions indicated:

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

7.1.4 \$set

Usage: **\$set(name,value)**

Category: assignment

Description:

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

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

Example:

The following statements will return the values indicated:

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

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

7.1.5 \$setmulti

Usage: **\$setmulti(name,value[,separator])**

Category: assignment

Implemented: Picard 1.0

Description:

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

Example:

The following statements will return the values indicated:

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

7.1.6 \$unset

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

Category: assignment

Description:

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

Example:

The following would unset all performer tags:

<code>\$unset (performer:*)</code>

7.2 Text Functions

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

7.2.1 \$delprefix

Usage: **\$delprefix(text[,prefixes])**

Category: text

Implemented: Picard 1.3

Description:

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

Example:

The following statements will return the values indicated:

<code>\$delprefix(The Beatles)</code>	<code>==></code>	"Beatles"
<code>\$delprefix(The Beatles,)</code>	<code>==></code>	"The Beatles"
<code>\$delprefix(THE Beatles)</code>	<code>==></code>	"THE Beatles"

(continues on next page)

(continued from previous page)

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

7.2.2 \$find

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

Category: text

Implemented: Picard 2.3

Description:

Returns the zero-based index of the first 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 that prior to Picard 2.3.2 `$find` returned "-1" if `needle` was not found.

Example:

The following statements will return the values indicated:

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

7.2.3 \$firstalphachar

Usage: **\$firstalphachar(text[,nonalpha])**

Category: text

Implemented: Picard 0.12

Description:

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

Example:

The following statements will return the values indicated:

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

7.2.4 \$firstwords

Usage: **\$firstwords(text,length)**

Category: text

Implemented: Picard 0.12

Description:

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

Example:

The following statements will return the values indicated:

<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>

7.2.5 \$get

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

Category: text

Description:

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

Example:

The following statements will return the values indicated:

```

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

```

7.2.6 \$initials

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

Category: text

Implemented: Picard 0.12

Description:

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

Example:

The following statements will return the values indicated:

```

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

```

7.2.7 \$left

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

Category: text

Description:

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

Example:

The following statements will return the values indicated:

```

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

```

(continues on next page)

(continued from previous page)

```
$left(ABC,-2) ==> "A"  
$left(ABC,-4) ==> ""
```

7.2.8 \$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()  
$len(ABC) ==> "3"
```

7.2.9 \$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"
```

7.2.10 \$num

Usage: **\$num(num,len)**

Category: text

Description:

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

Example:

The following statements will return the values indicated:

<code>\$num(,)</code>	<code>==></code>	<code>" "</code>
<code>\$num(,1)</code>	<code>==></code>	<code>"0"</code>
<code>\$num(a,)</code>	<code>==></code>	<code>" "</code>
<code>\$num(a,5)</code>	<code>==></code>	<code>"00000"</code>
<code>\$num(123,5)</code>	<code>==></code>	<code>"00123"</code>
<code>\$num(1.23,5)</code>	<code>==></code>	<code>"00000"</code>
<code>\$num(123,)</code>	<code>==></code>	<code>" "</code>
<code>\$num(123,0)</code>	<code>==></code>	<code>"123"</code>
<code>\$num(123,1)</code>	<code>==></code>	<code>"123"</code>
<code>\$num(123,20)</code>	<code>==></code>	<code>"00000000000000000000123"</code>
<code>\$num(123,50)</code>	<code>==></code>	<code>"00000000000000000000123"</code>
<code>\$num(123,5.5)</code>	<code>==></code>	<code>" "</code>
<code>\$num(1.23,10)</code>	<code>==></code>	<code>"0000000000"</code>

7.2.11 \$pad

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

Category: text

Description:

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

Example:

The following statements will return the values indicated:

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

7.2.12 \$replace

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

Category: text

Description:

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

Example:

The following statements will return the values indicated:

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

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

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

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

7.2.13 \$reverse

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

Category: text

Description:

Returns `text` in reverse order.

Example:

The following statements will return the values indicated:

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

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


7.2.14 \$right

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

Category: text

Description:

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

Example:

The following statements will return the values indicated:

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

7.2.15 \$rreplace

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

Category: text

Description:

Regular expression replace. This function will replace the matching group specified by `pattern` with `replace` in `text`. For more information about regular expressions, please see the [article on Wikipedia](https://wikipedia.org/wiki/Regular_expression)⁴⁸.

Example:

The following statements will return the values indicated:

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

⁴⁸ https://wikipedia.org/wiki/Regular_expression

7.2.16 \$rsearch

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

Category: text

Description:

Regular expression search. This function will return the first matching group specified by `pattern` from `text`. For more information about regular expressions, please see the [article on Wikipedia](https://wikipedia.org/wiki/Regular_expression)⁴⁹.

Example:

The following statements will return the values indicated:

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

7.2.17 \$strip

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

Category: text

Description:

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

Example:

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

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

⁴⁹ https://wikipedia.org/wiki/Regular_expression

7.2.18 \$substr

Usage: **\$substr(text,start,end)**

Category: text

Implemented: Picard 2.3

Description:

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

Example:

The following statements will return the values indicated:

<code>\$substr(abcdefg)</code>	<code>==></code>	<code>"abcdefg"</code>
<code>\$substr(abcdefg, 3)</code>	<code>==></code>	<code>"defg"</code>
<code>\$substr(abcdefg, , 3)</code>	<code>==></code>	<code>"abc"</code>
<code>\$substr(abcdefg, 0, 3)</code>	<code>==></code>	<code>"abc"</code>
<code>\$substr(abcdefg, -3)</code>	<code>==></code>	<code>"efg"</code>
<code>\$substr(abcdefg, -6, 3)</code>	<code>==></code>	<code>"bc"</code>
<code>\$substr(abcdefg, -10, 3)</code>	<code>==></code>	<code>"abc"</code>
<code>\$substr(abcdefg, 3, 1)</code>	<code>==></code>	<code>" "</code>

7.2.19 \$swapprefix

Usage: **\$swapprefix(text[,prefixes])**

Category: text

Implemented: Picard 1.3 (previously as a plugin since Picard 0.13)

Description:

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

Example:

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

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

(continues on next page)

(continued from previous page)

<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>

7.2.20 \$title

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

Category: text

Implemented: Picard 2.1

Description:

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

Example:

The following statement will return the value indicated:

<code>\$title(tHe houR is upOn uS)</code>	<code>==></code>	<code>"The Hour Is Upon Us"</code>
---	---------------------	------------------------------------

7.2.21 \$trim

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

Category: text

Description:

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

Examples:

The following statements will return the values indicated:

<code>\$trim(Trimmed)</code>	<code>==></code>	<code>"Trimmed"</code>
<code>\$trim(__Trimmed__,_)</code>	<code>==></code>	<code>"Trimmed"</code>
<code>\$trim(x__Trimmed__y,_x)</code>	<code>==></code>	<code>"Trimmed__y"</code>

7.2.22 \$truncate

Usage: **\$truncate(text,length)**

Category: text

Implemented: Picard 0.12

Description:

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

Example:

The following statements will return the values indicated:

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

7.2.23 \$upper

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

Category: text

Description:

Returns `text` in upper case.

Example:

The following statement will return the value indicated:

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

7.3 Multi-Value Functions

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

7.3.1 \$getmulti

Usage: **\$getmulti(name,index[,separator])**

Category: multi-value

Implemented: Picard 2.3

Description:

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

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

Example:

The following statements will return the values indicated:

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

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

7.3.2 \$join

Usage: **\$join(name,text[,separator])**

Category: multi-value

Implemented: Picard 2.3

Description:

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

Example:

The following statements will return the values indicated:

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

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

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

7.3.3 \$lenmulti

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

Category: multi-value

Description:

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

Example:

The following statements will return the values indicated:

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

(continues on next page)

(continued from previous page)

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

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

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

7.3.4 \$map

Usage: **\$map(name,code[,separator])**

Category: multi-value

Implemented: Picard 2.3

Description:

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

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

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

Example:

The following statements will return the values indicated:

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

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

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


7.3.5 \$performer

Usage: **\$performer(pattern[,separator])**

Category: multi-value

Implemented: Picard 0.10

Description:

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

Example:

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

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

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

7.3.6 \$reversemulti

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

Category: multi-value

Implemented: Picard 2.3.1

Description:

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

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

Example:

The following statements will return the values indicated:

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

$setmulti(bar,A; B; C; D; E)
```

(continues on next page)

(continued from previous page)

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

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

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

7.3.7 \$slice

Usage: **\$slice(name,start,end[,separator])**

Category: multi-value

Implemented: Picard 2.3

Description:

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

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

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

Example:

The following statements will return the values indicated:

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

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

$slice(A; B; C; D; E,1) ==> "B; C; D; E"
$slice(A; B; C; D; E,1,3) ==> "B; C"
$slice(A; B; C; D; E,,3) ==> "A; B; C"
$slice(A; B; C; D; E,1,3) ==> "B; C"
$slice(A; B; C; D; E,1,-1) ==> "B; C; D"
$slice(A; B; C; D; E,-4,4) ==> "B; C; D"
$slice(A:A; B:B; C:C; D:D; E:E,,1,:) ==> "A"
$slice(A:A; B:B; C:C; D:D; E:E,-2,, :) ==> "D; E:E"
$slice(A:A; B:B; C:C; D:D; E:E,2,4,:) ==> "B; C:C; D"
```

7.3.8 \$sortmulti

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

Category: multi-value

Implemented: Picard 2.3.1

Description:

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

Example:

The following statements will return the values indicated:

<code>\$set(foo,B; C; E; D; A)</code>	
<code>\$sortmulti(%foo%)</code>	<code>==> "B; C; E; D; A"</code>
 <code>\$setmulti(foo,B; C; E; D; A)</code>	
<code>\$sortmulti(%foo%)</code>	<code>==> "A; B; C; D; E"</code>
 <code>\$sortmulti(B; D; E; A; C)</code>	<code>==> "A; B; C; D; E"</code>
<code>\$sortmulti(B:AB; D:C; E:D; A:A; C:X,.)</code>	<code>==> "A; C:AB; D:B:C; E:D; A:X"</code>
<code>\$sortmulti(,)</code>	<code>==> ""</code>
<code>\$sortmulti(,:)</code>	<code>==> ""</code>

7.4 Mathematical Functions

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

7.4.1 \$add

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

Category: mathematical

Description:

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

Example:

The following statements will return the values indicated:

\$add(20, 15)	==>	"45"
\$add(20, -15)	==>	"5"
\$add(20, 14, 1)	==>	"35"
\$add(20, 10, 3, 2)	==>	"35"
\$add(20, 10, 3,)	==>	" "
\$add(20, 10, 3, a)	==>	" "
\$add(20, 10, 3.5)	==>	" "

7.4.2 \$div

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

Category: mathematical

Description:

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

Example:

The following statements will return the values indicated:

\$div(10, 3)	==>	"3"
\$div(10, -3)	==>	"-4"
\$div(-10, 3)	==>	"-4"
\$div(10, 3, 2)	==>	"1"
\$div(10, -3, -2)	==>	"2"
\$div(10, 2, 1.5)	==>	" "
\$div(10, 2, 0)	==>	" "
\$div(10, 2, x)	==>	" "
\$div(10, 2,)	==>	" "

7.4.3 \$mod

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

Category: mathematical

Description:

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

Example:

The following statements will return the values indicated:

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

7.4.4 \$mul

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

Category: mathematical

Description:

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

Example:

The following statements will return the values indicated:

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

7.4.5 \$sub

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

Category: mathematical

Description:

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

Example:

The following statements will return the values indicated:

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

7.5 Conditional Functions

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

7.5.1 \$and

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

Category: conditional

Description:

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

Example:

The following statements will return the values indicated:

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

7.5.2 \$endswith

Usage: **\$endswith(text,suffix)**

Category: conditional

Implemented: Picard 1.4

Description:

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

Example:

The statements below return the values indicated:

<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>

7.5.3 \$eq

Usage: **\$eq(x,y)**

Category: conditional

Implemented: Picard

Description:

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

Example:

The following statements will return the values indicated:

<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>

7.5.4 \$eq_all

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

Category: conditional

Description:

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

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

Example:

The following statements will return the values indicated:

<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>

7.5.5 \$eq_any

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

Category: conditional

Description:

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

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

Example:

The following statements will return the values indicated:

<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>

7.5.6 \$gt

Usage: **\$gt(x,y)**

Category: conditional

Implemented: Picard

Description:

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

Example:

The following statements will return the values indicated:

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

7.5.7 \$gte

Usage: **\$gte(x,y)**

Category: conditional

Implemented: Picard

Description:

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

Example:

The following statements will return the values indicated:

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

7.5.8 \$if

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

Category: conditional

Description:

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

Example:

The following statements will return the values indicated:

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

7.5.9 \$if2

Usage: **\$if2(a1,a2,a3,...)**

Category: conditional

Description:

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

Example:

The following statements will return the values indicated:

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

7.5.10 \$in

Usage: **\$in(x,y)**

Category: conditional

Implemented: Picard

Description:

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

Example:

The following statements will return the values indicated:

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

7.5.11 \$inmulti

Usage: **\$inmulti(%x%,y)**

Category: conditional

Implemented: Picard 1.0

Description:

Returns true if multi-value variable x contains exactly y as one of its values.

Note that comparisons are case-sensitive.

Example:

The following statements will return the values indicated:

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

7.5.12 \$is_audio

Usage: **\$is_audio()**

Category: conditional

Implemented: Picard 2.2

Description:

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

Example:

The following statements will return the values indicated:

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

7.5.13 \$is_complete

Usage: **\$is_complete()**

Category: conditional

Description:

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

Note that this only works in File Naming scripts.

Example:

The following statements will return the values indicated:

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

7.5.14 \$is_video

Usage: **\$is_video()**

Category: conditional

Implemented: Picard 2.2

Description:

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

Example:

The following statements will return the values indicated:

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

7.5.15 \$lt

Usage: **\$lt(x,y)**

Category: conditional

Description:

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

Example:

The following statements will return the values indicated:

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

7.5.16 \$lte

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

Category: conditional

Description:

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

Example:

The following statements will return the values indicated:

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

(continues on next page)

(continued from previous page)

<code>\$lte(a, 1)</code>	<code>==></code>	<code>" "</code>	<code>(False)</code>
<code>\$lte(1, a)</code>	<code>==></code>	<code>" "</code>	<code>(False)</code>
<code>\$lte(1, 1.5)</code>	<code>==></code>	<code>" "</code>	<code>(False)</code>

7.5.17 `$ne`

Usage: `$ne(x,y)`

Category: conditional

Description:

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

Example:

The following statements will return the values indicated:

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

7.5.18 `$ne_all`

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

Category: conditional

Description:

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

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

Example:

The following statements will return the values indicated:

<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>

7.5.19 \$ne_any

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

Category: conditional

Description:

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

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

Example:

The following statements will return the values indicated:

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

7.5.20 \$not

Usage: **\$not(x)**

Category: conditional

Description:

Returns true if x is empty.

Example:

The following statements will return the values indicated:

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

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

7.5.21 \$or

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

Category: conditional

Description:

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

Example:

The following statements will return the values indicated:

<code>\$or(,)</code>	<code>==></code>	<code>" "</code>	<code>(False)</code>
<code>\$or(,1)</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>
<code>\$or(,A)</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>
<code>\$or(,\$gt(4,5))</code>	<code>==></code>	<code>" "</code>	<code>(False)</code>
<code>\$or(,\$lt(4,5))</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>
<code>\$or(,,)</code>	<code>==></code>	<code>" "</code>	<code>(False)</code>
<code>\$or(,,0)</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>
<code>\$or(,,)</code>	<code>==></code>	<code>"1"</code>	<code>(True)</code>

7.5.22 \$startswith

Usage: **\$startswith(text,prefix)**

Category: conditional

Implemented: Picard 1.4

Description:

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

Example:

The statements below return the values indicated:

<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>

7.6 Information Functions

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

7.6.1 \$datetime

Usage: **\$datetime([format])**

Category: information

Implemented: Picard 2.3

Description:

Returns the current date and time in the specified format, which is based on the standard Python `strftime` [format codes](#)⁵⁰. If no format is specified the date and time will be returned in the form '2020-02-05 14:26:32'. Note that any special characters such as '%', '\$', '(', ')' and " will need to be escaped as shown in the examples below.

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

Example:

The following statements will return the values indicated:

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

7.6.2 \$matchedtracks

Usage: **\$matchedtracks()**

Category: information

Implemented: Picard 0.12

Description:

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

Example:

The following statements will return the values indicated:

⁵⁰ <https://strftime.org>

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

7.7 Loop Functions

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

7.7.1 \$foreach

Usage: **\$foreach(name,code,separator=”; “)**

Category: loop

Implemented: Picard 2.3

Description:

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

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

Example:

The following statements will perform the processing indicated:

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

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

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

7.7.2 \$while

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

Category: loop

Implemented: Picard 2.3

Description:

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

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

Example:

The following statement will set `return` to “Echo... echo... echo...”:

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

7.8 Miscellaneous Functions

The miscellaneous scripting functions are:

7.8.1 \$noop

Usage: **\$noop(...)**

Category: miscellaneous

Description:

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

Example:

The following statements will return the values indicated:

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

USING PICARD

This section is still under development.

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.

9.1 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](#)⁵¹ that have been submitted to the MusicBrainz Picard repository shown on the Picard website.

9.2 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](#)⁵², and there is a thread specific to [file naming and script snippets](#)⁵³.

9.2.1 File Naming Script

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

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

⁵¹ <https://picard.musicbrainz.org/plugins/>

⁵² <https://community.metabrainz.org/>

⁵³ <https://community.metabrainz.org/t/repository-for-neat-file-name-string-patterns-and-tagger-script-snippets/2786/>

9.2.2 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.

9.3 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.

9.3.1 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

9.3.2 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 that 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.

9.3.3 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.

9.3.4 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.

9.3.5 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.

9.3.6 Removing Albums

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

9.3.7 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.

EXAMPLES

This section is still under development.

APPENDICES

11.1 Plugins API

11.1.1 Plugin Metadata

Each plugin must provide some metadata as variables. Those variables should be placed at the top of the file.

```
PLUGIN_NAME = "Example plugin"
PLUGIN_AUTHOR = "This authors name"
PLUGIN_DESCRIPTION = "This plugin is an example"
PLUGIN_VERSION = '0.1'
PLUGIN_API_VERSIONS = ['2.1', '2.2']
PLUGIN_LICENSE = "GPL-2.0-or-later"
PLUGIN_LICENSE_URL = "https://www.gnu.org/licenses/gpl-2.0.html"
```

Variables explanation:

- **PLUGIN_DESCRIPTION** should be as simple as possible, while still describing the main function.
- **PLUGIN_VERSION** should be filled with the version of Plugin.
- **PLUGIN_API_VERSIONS** should be set to the versions of Picard this plugin to run with. New Picard versions will usually support older plugin API versions, but on breaking changes support for older plugin versions can be dropped. Versions available for Picard 2 are “2.0”, “2.1” and “2.2”.
- **PLUGIN_LICENSE** should be set with the license name of the plugin. If possible use one of the license names from the [SPDX License List](https://spdx.org/licenses/)⁵⁴, 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.

⁵⁴ <https://spdx.org/licenses/>

11.1.2 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)
```

11.1.3 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 are available since API version 2.2*

11.1.4 File Formats

Plugins can extend Picard with support for additional file formats. See the existing [file format implementations](#)⁵⁵ 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()
```

(continues on next page)

⁵⁵ <https://github.com/metabrainz/picard/tree/master/picard/formats>

(continued from previous page)

```

    # 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)

```

11.1.5 Tagger Script Functions

To define new tagger script functions use `register_script_function(function, name=None)` from the `picard.script` module. `parser` is an instance of `picard.script.ScriptParser`, and the rest of the arguments passed to it are the arguments from the function call in the tagger script. Example:

```

PLUGIN_NAME = "Initials"
PLUGIN_AUTHOR = "Lukas Lalinsky"
PLUGIN_DESCRIPTION = "Provides tagger script function $initials(text)."
PLUGIN_VERSION = '0.1'
PLUGIN_API_VERSIONS = ['2.0']
PLUGIN_LICENSE = "GPL-2.0"
PLUGIN_LICENSE_URL = "https://www.gnu.org/licenses/gpl-2.0.txt"

from picard.script import register_script_function

def initials(parser, text):
    return "".join(a[:1] for a in text.split(" ") if a[:1].isalpha())

register_script_function(initials)

```

`register_script_function` supports two optional arguments:

- **eval_args**: If this is **False**, the arguments will not be evaluated before being passed to **function**.
- **check_argcount**: If this is **False** the number of arguments passed to the function will not be verified.

The default value for both arguments is **True**.

11.1.6 Context Menu Actions

Right-click context menu actions can be added to albums, tracks and files in “Unmatched Files”, “Clusters” and the “ClusterList” (parent folder of Clusters). Example:

```
PLUGIN_NAME = u'Remove Perfect Albums'
PLUGIN_AUTHOR = u'ichneumon, hrglgrmpf'
PLUGIN_DESCRIPTION = u'''Remove all perfectly matched albums from the
↳selection.'''
PLUGIN_VERSION = '0.2'
PLUGIN_API_VERSIONS = ['0.15.1']
PLUGIN_LICENSE = "GPL-2.0"
PLUGIN_LICENSE_URL = "https://www.gnu.org/licenses/gpl-2.0.txt"

from picard.album import Album
from picard.ui.itemviews import BaseAction, register_album_action

class RemovePerfectAlbums(BaseAction):
    NAME = 'Remove perfect albums'

    def callback(self, objs):
        for album in objs:
            if isinstance(album, Album) and album.is_complete() \
                and album.get_num_unmatched_files() == 0 \
                and album.get_num_matched_tracks() == len(list(album.
↳iterfiles())) \
                and album.get_num_unsaved_files() == 0 and album.loaded ==
↳True:
                    self.tagger.remove_album(album)

register_album_action(RemovePerfectAlbums())
```

Use `register_x_action` where ‘x’ is “album”, “track”, “file”, “cluster” or “clusterlist”.