PyQModManager Documentation

Release 1.0.0-alpha12

bicobus

Sep 02, 2020

Contents:

1 User's Guide **3**
- 1.1 PyQModManager . . . 3
- 1.2 Usage . . . 4
- 1.3 Version History . . . 6
- 1.4 qmm . . . 8

2 Contribute **17**

3 Indices and tables **19**

Python Module Index **21**

Index **23**

A kind of archive manager for easy handling of game modules.

Works on Python 3.7+.

CHAPTER 1

User's Guide

1.1 PyQModManager

Simple tool to manage a set of archives. Written to manage mods available for the game Lilith's Throne.

- Track the state of the different files bundled with each archives
- Unpack into a designated directory

1.1.1 Installation and Requirements

Windows

If you are running windows, a self contained binary is provided at each release. Please check the releases page to download the archive. Running the application is a simple as double clicking the .exe file present in the archive.

Linux

You'll need to install python3 with whatever package manager your distribution provides you. This application has been developped using python 3.7, any prior version might work, but untested. If you feel adventurous, you can checkout pyenv.

This software being a python script, it doesn't need to be installed. However requirements are needed for that purpose.

You'll need whatever is presesent in the requirements.txt. As Linux distributions such as debian are commonly out of date, I'd recommend installing the requirement locally, under your user directory, with the following command.

```
~$ pip install --user -r requirements.txt
```

You could also opt to install in a virtual environment through pipenv. `pipenv install` will use the Pipfile already present in the repository.

```
~$ pip install pipenv
~$ pipenv install
```

Running the app

If you chose pipenv, you can then start the application using the following, provided you are in the same folder as the run.py file.

```
~$ pipenv run ./run.py
```

If you opted for the simple pip install, simply execute run.py.

```
~$ python run.py
```

1.1.2 Known issues

The software is currently being rewritten, as such no known limitation exists. This might change once we get out of alpha.

1.1.3 Hacking

If you want to hack around, you only require the dependencies listed in the requirements.txt file. The Pipfile has a list of dev-packages, which is unneeded to actually run and develop the software. They're helper tools, like pylint or flake8

Documentation generator

The documentation is currently available at https://qmodmanager.rtfd.io/

The generation of the documentation on readthedocs.org necessitate some extra steps in order to successfuly generate the api documentation. We have to generate ui files at build time, which are not included in the repository nor available to rtd. Therefore a script apidoc has been provided as helper.

The apidoc script will forcefully generate the required files for the api. In addition of that, it will also parse the various UI files present in the resources folder and generate stubs in the _ext folder. Those stubs needs to be regenerated whenever a UI file is added to the resources folder.

1.2 Usage

In this document will inform you about various elements of the graphical user interface of PyQModManager.

1.2.1 Settings

The software needs to know two on disk location: where the game is located and a space to store the modules you've downloaded.

The game location should be the folder containing the .jar or .exe of the game. The repository for your module should be a random *empty* folder of your choice.

At the time I write these lines, the settings window is still a WiP.

1.2.2 Adding modules to be tracked by the software

The software keeps track of 3 types of archives: Rar files, 7z files and Zip files. Two ways exists to have the game track modules:

1. Drop an archive in the repository folder.
2. Use the button from the toolbar.

If you use , the archive file will be copied over the repository folder leaving the original file untouched.

1.2.3 Removing an archive

Select the archive you wish to delete and click on the trashbin () button. Alternatively, right-click on the archive and select the appropriate option. PyQModManager sends all removed archives to your trashbin, which you will need to empty manually.

1.2.4 Installing a module

Select the archive you wish to install then click on the install () button of the toolbar. Alternatively, you can right-click the archive and select the install option.

Only files natively handled by Lilith's Throne mod system are supported. Any other file or folder will be ignored.

Supported paths structure

A proper structure for a mod would be the following:

```
namespace
|-- items
|   |-- clothing
|   |   `-- ItemName
|   |       |-- cloth_test.svg
|   |       `-- cloth_test.xml
|   |-- tattoos
|   |   `-- ItemName
|   |       |-- tattoo_test.svg
|   |       `-- tattoo_test.xml
|   `-- weapons
|       `-- ItemName
|           |-- weapon_test.svg
|           `-- weapon_test.xml
`-- outfits
    `-- OutfitName
        `-- outfit_test.xml
```

The first folder must be the namespace, or colloquially the name of the modder. The module will be ignored if it is packaged with the initial `/res/mods` folders. The software will look for the existance of the subfolders `items`, `clothing`, `weapons`, `tattoos` and `outfits`.

1.2.5 Uninstalling a module

Select the archive you wish to uninstall then click on the uninstall () button of the toolbar. Alternatively, you can right-click the archive and select the uninstall option.

1.2.6 Monitoring of the hard drive

The software will monitor filesystem changes on both the game's module folder and the folder you designated as repository. The software will automatically scan any archive dropped in your repository folder, as well as making sure the game's module folder remains known even if you unpack files through other means than PyQModManager.

You can toggle off that behavior through the auto-refresh checkbox located above the list of your available modules. Doing so will activate a refresh button, located right next to the checkbox, which will allow you to manually refresh the internal database if you make changes to the file system.

The monitoring of the filesystem is designed to be as lightweight as possible. It disable itself whenever PyQModManager becomes inactive (alt-tab or minimized), and reactive itself whenever the software gain focus. Gaining back focus will force a refresh of the database on a needed basis: if nothing has changed, nothing is done.

1.3 Version History

1.3.1 1.0.0-alpha12 - ????????

Added

- A context menu on the treeview if the file is present on disk:
  - Open containing folder
  - Open file using text editor, graphics editor or both (for svg)
- List untracked files present in the `res/mods` folder. It is understood by untracked that files existing in the folder weren't found in any of the archives.
- Support for new mod files
  - `res/mods/statusEffects`
  - `res/mods/setBonuses`
  - `res/mods/items/items`

Changed

- Directories in the treeview now properly show their status.
- Context menus rewritten in a less stupid way.
- Archives context menu disable entries when they don't apply, an archive that is not installed cannot be uninstalled and so on.

- Got rid of the resources files for the setting window. It is now programatically built, which helps with maintenance.

1.3.2 1.0.0-alpha11 - 20-05-12

Added

- Color code each managed item based on their status
  - Each line has a dual color: left and right
  - Right side can either be transparent or red, to show existing conflicts.
  - Left side can either be green, blue or yellow
    - Yellow is for missing files
    - Blue is for mismatched files
    - Green is when every files of the archive matches on the drive.
  - Greyed out text means the archive contains nothing that can be installed
  - The Help buttons will send users to the readthedocs website.

Changed

- Each file is now beautifully displayed in a tree instead of using a TextInput
- Files are color coded depending on their states.
- The conflicts tab details where a file as been found as duplicate: *GameFile* or *Archive*

Fixed

- Fix crash related to file system watch (watchdog)

1.3.3 1.0.0-alpha10

- Same as alpha9, but working.

1.3.4 1.0.0-alpha9

- Send archives to the trashbin instead of a full removal from the hard drive.
- Foundations for the internationalisation (l10n) of the software through gettext
- A Watchdog to monitor both the module's repository and the game's module path
  - The software will automatically add whatever archive dropped in the module's repository
  - The software will automatically determine if the game's module directory has been modified and regenerate it's database the next time the application gain focus
  - A checkbox exists to disable this behavior if unchecked.
- Internal dev stuff: changes of libraries used, reworking codebase, etc

1.4 qmm

1.4.1 qmm package

qmm.**get_base_path**()

qmm.**get_data_path**(*relpath*)

qmm.**is_frozen**()

Submodules

qmm.bucket module

Buckets of dicts with a set of helpers function.

This module serves has a stand-in database, any function or method it contain would be facilitator to either access or transform the data. This module is necessary in order to keep track of the state of the different files and make that specific state available globally within the other modules.

class qmm.bucket.**FileMetadata**(*crc, path: Union[str, pathlib.Path], attributes, modified, isfrom*)

Bases: object

Representation of a file.

Can handle game files, mod files or file information comming from an archive.

Parameters

- **crc** (*int*) – CRC32 of the represented file, 0 or empty if file is a folder.
- **path** (*str or os.PathLike*) – relative path to the represented file.
- **attributes** (*str or None*) – 'D' for folder, 'F' otherwise. If the value passed is None, the attributes will be deduced from *path*
- **modified** (*str or None*) – timestamp of the last modification of the file.
- **isfrom** (*int or str*) – Will be the name of the archive the file originates from. Otherwise either *TYPE_GAMEFILE* or *TYPE_LOOSEFILE*.

as_dict()

Return this object as a dict (kinda).

attributes

crc

exists()

Check if the file exists on the disk.

is_dir()

Check if the represented item is a directory.

is_file()

Check if the represented item is a file.

modified

origin

path

path_as_posix()
Return 'pathlib.PurePosixPath' with self._Path as value.

split()

qmm.bucket.**TYPE_GAMEFILE = 2**
File present in an archive

qmm.bucket.**TYPE_LOOSEFILE = 1**
File present on the disk

qmm.bucket.**as_conflict**(*key: str*, *value*)
Append and item to the conflicts bucket

qmm.bucket.**as_gamefile**(*crc: Crc32, value: Union[pathlib.Path, pathlib.PurePath]*)
Add to the gamefiles a path indexed to its target CRC32.

qmm.bucket.**as_loosefile**(*crc: Crc32*, *filepath: pathlib.Path*)
Adds filepath to the loosefiles bucket, indexed on given CRC.

qmm.bucket.**file_crc_in_loosefiles**(*filemd: qmm.bucket.FileMetadata*) → bool
Check if a file's crc exists in loosefile's index.

qmm.bucket.**file_path_in_loosefiles**(*filemd: qmm.bucket.FileMetadata*) → bool
Check if a file's path exists within the different loosefile lists.

qmm.bucket.**remove_item_from_loosefiles**(*file: qmm.bucket.FileMetadata*)
Removes the reference to file if it is found in loosefiles.

qmm.bucket.**with_conflict**(*path: str*) → bool
Check if path exists in conflicts's keys.

The conflicts bucket purpose is to list issues in-between archives only.

Parameters **path** (*str*) – Simple string, should be a path pointing to a file

Returns True if path exist in conflicts's keys

Return type bool

qmm.bucket.**with_gamefiles**(*crc: Crc32 = None*, *path: str = None*)
Determine if a file exists within the cached list of game files.

First check if a CRC32 exist within the gamefiles bucket, if no CRC is given or the check fails, will then check if a path is present in the gamefiles's bucket values.

Parameters

- **crc** (*int*) – CRC32 as integer
- **path** (*str*) – the relative pathlike string of a file

Returns True if either CRC32 or path are found

Return type bool

qmm.common module

qmm.common.**acommand**(*alias*)

qmm.common.**bundled_tools_path**()
Returns the path to the bundled 7z executable.

qmm.common.**command**(*binary*, *alias=False*)
Return path to binary or None if not found.

Analogous to bash's command, but do not actually execute anything.

Parameters

- **binary** (*str*) – Name of binary to find in PATH
- **alias** (*bool*) – True if the name is an alias to be looked up the pre-made dict. *alias* is only useful for windows OS as the binary can be a tuple. Aliases are also used to find predefined software without the need of calling them by name, good for cross platform.

Returns Path to the binary or None if not found.

Return type os.Pathlike or None

qmm.common.**settings = <qmm.config.Config object>**
instance of the Config object that governs the user's preferences. Can be imported anywhere in the app

qmm.common.**settings_are_set**()
Returns False if either 'local_repository' or 'game_folder' isn't set.

qmm.common.**startfile**(*file*)

qmm.common.**timestamp_to_string**(*timestamp*)
Takes a UNIX timestamp and return a vernacular date.

qmm.common.**valid_suffixes**(*output_format='qfiledialog'*) → Union[List[str], Tuple[str, str, str], bool]
Properly format a list of filters for QFileDialog.

Parameters **output_format** (*str*) – Accepts either 'qfiledialog' or 'pathlib'. 'pathlib' returns a simple list of suffixes, whereas 'qfiledialog' format the output to be an acceptable filter for QFileDialog.

Returns a list of valid suffixes.

Return type list

qmm.config module

class qmm.config.**Config**(*filename*, *config_dir=None*, *defaults=None*, *compress=False*)
Bases: collections.abc.MutableMapping

Influenced by deluge's config object.

delayed_save(*msec=5000*)
Schedule a save in the future if one isn't already planned.

load(*filename=None*)

save(*filename=None*)

exception qmm.config.**SettingsNotSetError**
Bases: Exception

qmm.config.**get_config_dir**(*filename=None*, *extra_directories=None*) → str
Return the full path of the user config dir.

Parameters

- **filename** – If provided, gets added at the end of the string.
- **extra_directories** – If provided, extends on the returned path.

qmm.dialogs module

Contains a bunch of helper function to display Qt's dialogs.

class qmm.dialogs.**SplashProgress**(*parent*, *title*, *message*)
Bases: QDialog, qmm.ui_qprogress.Ui_Dialog

progress(*text: str*, *category: str = None*)

qmm.dialogs.**qError**(*message*, ***kwargs*)
Helper function to show an error dialog.

qmm.dialogs.**qInformation**(*message*, ***kwargs*)
Helper function to show an informational dialog.

qmm.dialogs.**qWarning**(*message*, ***kwargs*)
Helper function to show a warning dialog.

qmm.dialogs.**qWarningYesNo**(*message*, ***kwargs*)
Helper function to show an Y/N warning dialog.

qmm.filehandler module

exception qmm.filehandler.**ArchiveException**
Bases: Exception

class qmm.filehandler.**ArchiveInstance**(*archive_name*, *file_list*)
Bases: qmm.ab.archives.ABCArchiveInstance

Represent an archive and its content already analyzed and ready for display.

ar_type = 1

conflicts()
Yield FileMetadata of conflicting entries of the archive.

ignored()
Yield file metadata of ignored entries of the archive.

install_info()
Return a several lists useful to the installation process.

The content in matched and ignored key will be compiled into a set of exclude flags, whereas the content of mismatched key will be overridden.

See also:

install_archive()

matched()
Yield file metadata of matched entries of the archive.

mismatched()
Yield file metadata of mismatched entries of the archive.

missing()
Yield file metadata of missing entries of the archive.

reset_conflicts()
Generate a list of conflicting files, either from in the game folders or in other archives, for each file present in this archive.

uninstall_info()
Informations necessary to the uninstall function.

class qmm.filehandler.**ArchivesCollection**
Bases: collections.abc.MutableMapping, typing.Generic

Manage sets of *ArchiveInstance*.

add_archive(*path*, *hashsum: str = None*, *progress=None*)
Add an archive to the list of managed archives.

This method should be used over __setitem__ as it setup the different metadata required by the UI.

build_archives_list(*progress*, *rebuild=False*)

diff_matched_with_loosefiles()

find(*archive_name: str = None*, *hashsum: str = None*)
Find a member based on the name or hashsum of the archive.

If archiveName is not None, will check if archiveName exists in the keys of the collection. If hashsum is not None, will check if the value exists in the self._hashsums dict. If all checks fails, returns False.

Parameters

- **archive_name** – filename of the archive, suffix included (default None)
- **hashsum** – sha256sum of the file (default None)

Returns Boolean or ArchiveInstance

hashsums(*key*)

initiate_conflicts_detection()

refresh() → Iterable[Tuple[int, str]]
Scan the local repository to add or remove archives as needed.

This is a companion method to use with WatchDog whenever something changes on the filesystem.

Yields (Union[ArchiveEvents.FILE_ADDED, ArchiveEvents.FILE_REMOVED], str) – State and name of the file.

rename_archive(*src_path*, *dest_path*)
Rename the key pointing to an archive.

Whenever an archive on the drive gets renamed, we need to do the same with the key under which the parsed data is stored.

special

stat(*key*)

class qmm.filehandler.**VirtualArchiveInstance**(*file_list*)
Bases: qmm.ab.archives.ABCArchiveInstance

ar_type = 2

conflicts()
Yield FileMetadata of conflicting entries of the archive.

ignored()
Yield file metadata of ignored entries of the archive.

install_info()

matched()
Yield file metadata of matched entries of the archive.

mismatched()
Yield file metadata of mismatched entries of the archive.

missing()
Yield file metadata of missing entries of the archive.

reset_conflicts()

uninstall_info()
Informations necessary to the uninstall function.

qmm.filehandler.**build_cmd**(*filepath*, **ext*, *extract=True*, *output=None*, ***extra*)

qmm.filehandler.**build_game_files_crc32**(*progress=None*)
Compute the CRC32 value of all the game files then add them to a bucket.

The paths returned by this function are non-existent due to a difference between the mods and the game folder structure. It is needed to be that way in order to compare the mod files with the existing game files.

Parameters **progress** (*progress()*) – Callback to a method accepting strings as argument.

qmm.filehandler.**build_loose_files_crc32**(*progress=None*)
Build the CRC32 value of all loose files.

Parameters **progress** (*progress()*) – Callback to a method accepting strings as argument.

qmm.filehandler.**conflicts_process_files**(*files*, *archives_list*, *current_archive*, *processed*)
Process an archive, verify that each of its files are unique.

Parameters

- **files** (files()) – Process the files fed by the instance method.
- **archives_list** (*ArchivesCollection*) – Instance of ArchivesCollection.
- **current_archive** (*str*) – Filename on the disk of the current archive being processed.
- **processed** (*list or None*) – List of processed archives. Set to None if only one archive needs to be processed.

qmm.filehandler.**copy_archive_to_repository**(*filename*)
Copy an archive to the manager's repository.

qmm.filehandler.**delete_archive**(*filepath*)
Delete an archive from the filesystem.

qmm.filehandler.**extract7z**(*file_archive: pathlib.Path*, *output_path: pathlib.Path*, *exclude_list=None*, *progress=None*) → Union[List[qmm.bucket.FileMetadata], bool]

qmm.filehandler.**file_in_other_archives**(*file: qmm.bucket.FileMetadata*, *archives: qmm.filehandler.ArchivesCollection*, *ignore: List[T]*) → List[T]
Search for existence of file in other archives.

Parameters

- **file** (FileMetadata) – file to be found
- **archives** (ArchivesCollection) – instance of ArchivesCollection
- **ignore** (*list*) – list of archives to ignore, for example already parsed archives

Returns List of archives containing the same file.

Return type List

qmm.filehandler.**generate_conflicts_between_archives**(*archives_lists: qmm.filehandler.ArchivesCollection*, *progress=None*)

qmm.filehandler.**get_mod_folder**(*with_file: str = None*, *prepend_modpath=False*) → pathlib.Path
Return the path to the game folder.

Parameters

- **with_file** – append 'with_file' to the path
- **prepend_modpath** – if True, adds the module path before 'with_file'

Returns PathLike structure representing the game folder.

qmm.filehandler.**install_archive**(*file_to_extract: str, file_context: Dict[str, List[qmm.bucket.FileMetadata]]*) → Union[bool, List[qmm.bucket.FileMetadata]]
Install the content of an archive into the game mod folder.

Parameters

- **file_to_extract** (*str*) – path to the archive to extract.
- **file_context** (*dict*) – A dict containing the keys *matched*, *mismatched*, *ignored*. Each of these entries point to a list containing *FileMetadata* objects.

  The content in matched and ignored key will be compiled into a set of exclude flags, whereas the content of mismatched key will be overridden. See *ArchiveInstance.install_info()*

Returns Output of function *extract7z()* or False

qmm.filehandler.**list7z**(*file_path: Union[str, pathlib.Path], progress=None*) → List[qmm.bucket.FileMetadata]

qmm.filehandler.**reErrorMatch**()
Matches zero or more characters at the beginning of the string.

qmm.filehandler.**reExtractMatch**()
Matches zero or more characters at the beginning of the string.

qmm.filehandler.**reListMatch**()
Matches zero or more characters at the beginning of the string.

qmm.filehandler.**sha256hash**(*filename: Union[IO, str]*) → Optional[str]
Return the 256 hash of the managed archive.

Parameters **filename** – path to the file to hash

Returns a string if successful, otherwise None

Return type str or None

qmm.filehandler.**uninstall_files**(*file_list: List[qmm.bucket.FileMetadata]*)
Removes a list of files and directory from the filesystem.

Parameters **file_list** (*list[FileMetadata]*) – A list of *FileMetadata* objects.

Returns True on success, False if an error occurred during the deleting process.

Return type bool

Notes

Any error will be logged silently to the application configured facility.

qmm.lang module

qmm.lang.**get_locale**()

qmm.lang.**list_available_languages**()

qmm.lang.**normalize_locale**(*loc: str*)

qmm.lang.**set_gettext**(*install=True*)

qmm.manager module

qmm.version module

qmm.widgets module

CHAPTER 2

Contribute

Did you find a bug or have a feature request for PyQModManager? You can file an issue ticket at the issue tracker. You can also ask questions at the Lilith's Throne official discord.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

q

qmm, 8
qmm.bucket, 8
qmm.common, 9
qmm.config, 10
qmm.dialogs, 11
qmm.filehandler, 11
qmm.lang, 15
qmm.version, 15

Index

A

acommand() (*in module qmm.common*), 9
add_archive() (*qmm.filehandler.ArchivesCollection method*), 12
ar_type (*qmm.filehandler.ArchiveInstance attribute*), 11
ar_type (*qmm.filehandler.VirtualArchiveInstance attribute*), 12
ArchiveException, 11
ArchiveInstance (*class in qmm.filehandler*), 11
ArchivesCollection (*class in qmm.filehandler*), 12
as_conflict() (*in module qmm.bucket*), 9
as_dict() (*qmm.bucket.FileMetadata method*), 8
as_gamefile() (*in module qmm.bucket*), 9
as_loosefile() (*in module qmm.bucket*), 9
attributes (*qmm.bucket.FileMetadata attribute*), 8

B

build_archives_list() (*qmm.filehandler.ArchivesCollection method*), 12
build_cmd() (*in module qmm.filehandler*), 13
build_game_files_crc32() (*in module qmm.filehandler*), 13
build_loose_files_crc32() (*in module qmm.filehandler*), 13
bundled_tools_path() (*in module qmm.common*), 9

C

command() (*in module qmm.common*), 9
Config (*class in qmm.config*), 10
conflicts() (*qmm.filehandler.ArchiveInstance method*), 11
conflicts() (*qmm.filehandler.VirtualArchiveInstance method*), 12
conflicts_process_files() (*in module qmm.filehandler*), 13
copy_archive_to_repository() (*in module qmm.filehandler*), 13
crc (*qmm.bucket.FileMetadata attribute*), 8

D

delayed_save() (*qmm.config.Config method*), 10
delete_archive() (*in module qmm.filehandler*), 13
diff_matched_with_loosefiles() (*qmm.filehandler.ArchivesCollection method*), 12

E

exists() (*qmm.bucket.FileMetadata method*), 8
extract7z() (*in module qmm.filehandler*), 13

F

file_crc_in_loosefiles() (*in module qmm.bucket*), 9
file_in_other_archives() (*in module qmm.filehandler*), 13
file_path_in_loosefiles() (*in module qmm.bucket*), 9
FileMetadata (*class in qmm.bucket*), 8
find() (*qmm.filehandler.ArchivesCollection method*), 12

G

generate_conflicts_between_archives() (*in module qmm.filehandler*), 14
get_base_path() (*in module qmm*), 8
get_config_dir() (*in module qmm.config*), 10
get_data_path() (*in module qmm*), 8
get_locale() (*in module qmm.lang*), 15
get_mod_folder() (*in module qmm.filehandler*), 14

H

hashsums() (*qmm.filehandler.ArchivesCollection method*), 12

I

ignored() (*qmm.filehandler.ArchiveInstance method*), 11

ignored() (*qmm.filehandler.VirtualArchiveInstance method*), 12
initiate_conflicts_detection() (*qmm.filehandler.ArchivesCollection method*), 12
install_archive() (*in module qmm.filehandler*), 14
install_info() (*qmm.filehandler.ArchiveInstance method*), 11
install_info() (*qmm.filehandler.VirtualArchiveInstance method*), 12
is_dir() (*qmm.bucket.FileMetadata method*), 8
is_file() (*qmm.bucket.FileMetadata method*), 8
is_frozen() (*in module qmm*), 8

L

list7z() (*in module qmm.filehandler*), 14
list_available_languages() (*in module qmm.lang*), 15
load() (*qmm.config.Config method*), 10

M

matched() (*qmm.filehandler.ArchiveInstance method*), 11
matched() (*qmm.filehandler.VirtualArchiveInstance method*), 12
mismatched() (*qmm.filehandler.ArchiveInstance method*), 11
mismatched() (*qmm.filehandler.VirtualArchiveInstance method*), 13
missing() (*qmm.filehandler.ArchiveInstance method*), 11
missing() (*qmm.filehandler.VirtualArchiveInstance method*), 13
modified (*qmm.bucket.FileMetadata attribute*), 8

N

normalize_locale() (*in module qmm.lang*), 15

O

origin (*qmm.bucket.FileMetadata attribute*), 8

P

path (*qmm.bucket.FileMetadata attribute*), 8
path_as_posix() (*qmm.bucket.FileMetadata method*), 8
progress() (*qmm.dialogs.SplashProgress method*), 11

Q

qError() (*in module qmm.dialogs*), 11
qInformation() (*in module qmm.dialogs*), 11
qmm (*module*), 8
qmm.bucket (*module*), 8
qmm.common (*module*), 9
qmm.config (*module*), 10
qmm.dialogs (*module*), 11
qmm.filehandler (*module*), 11
qmm.lang (*module*), 15
qmm.version (*module*), 15
qWarning() (*in module qmm.dialogs*), 11
qWarningYesNo() (*in module qmm.dialogs*), 11

R

reErrorMatch() (*in module qmm.filehandler*), 14
reExtractMatch() (*in module qmm.filehandler*), 14
refresh() (*qmm.filehandler.ArchivesCollection method*), 12
reListMatch() (*in module qmm.filehandler*), 14
remove_item_from_loosefiles() (*in module qmm.bucket*), 9
rename_archive() (*qmm.filehandler.ArchivesCollection method*), 12
reset_conflicts() (*qmm.filehandler.ArchiveInstance method*), 11
reset_conflicts() (*qmm.filehandler.VirtualArchiveInstance method*), 13

S

save() (*qmm.config.Config method*), 10
set_gettext() (*in module qmm.lang*), 15
settings (*in module qmm.common*), 10
settings_are_set() (*in module qmm.common*), 10
SettingsNotSetError, 10
sha256hash() (*in module qmm.filehandler*), 14
special (*qmm.filehandler.ArchivesCollection attribute*), 12
SplashProgress (*class in qmm.dialogs*), 11
split() (*qmm.bucket.FileMetadata method*), 9
startfile() (*in module qmm.common*), 10
stat() (*qmm.filehandler.ArchivesCollection method*), 12

T

timestamp_to_string() (*in module qmm.common*), 10
TYPE_GAMEFILE (*in module qmm.bucket*), 9
TYPE_LOOSEFILE (*in module qmm.bucket*), 9

U

uninstall_files() (*in module qmm.filehandler*), 14
uninstall_info() (*qmm.filehandler.ArchiveInstance method*), 11

uninstall_info() (*qmm.filehandler.VirtualArchiveInstance method*), 13

V

valid_suffixes() (*in module qmm.common*), 10
VirtualArchiveInstance (*class in qmm.filehandler*), 12

W

with_conflict() (*in module qmm.bucket*), 9
with_gamefiles() (*in module qmm.bucket*), 9