EM Media Handler Documentation

Release 1.0b2

Erin Morelli

February 07, 2015

Contents

Welcome to to the documentation for EM Media Handler, a comprehensive media automation system for Linux and OS X.

To get started, begin with the *Getting Started* guide. Which will walk you through the basics of installation and configuration.

Get more detailed information on advanced features in the *Configuration* section.

If you're a developer, check out the *For Developers* section on using or extending mediahandler for your projects.

If you run into any problems, please report an issue in the github issue tracker.

To stay up-to-date on project releases, please join our mailing list.

CHAPTER 1

Contents

1.1 Getting Started

Welcome to EM Media Handler! This guide will help you get started with automating your media.

1.1.1 Requirements

EM Media Handler supports Python <= 2.7 - there are plans to support 3.x in a future release.

Check out the *Installation Requirements* for details on required python packages and 3rd party applications.

Make sure your system is ready before proceeding.

1.1.2 Quick Installation

The easiest way to **install** is via *pip*:

```
pip install em-media-handler
```

To **upgrade** to the latest version:

```
pip install -U em-media-handler
```

1.1.3 Installing from Source

1. Download the source from either GitHub or BitBucket.
2. From inside the downloaded source folder, run the build command:

   ```
   sudo python setup.py build
   ```

   Note: If you have run the build before, ensure you have a clean build environment first by running: `sudo python setup.py clean -a`

3. Install the package:

   ```
   sudo python setup.py install
   ```

1.1.4 User Settings

The default user configuration is installed to:

```
~/.config/mediahandler/config.yml
```

Use any text editor to open and edit the file. Refer to the *User Settings* article more details on the settings available.

1.1.5 Usage

To get started type:

```
addmedia --help
```

to view the available options. Read more about the *Command-line Interface*.

It is also possible to integrate EM Media Handler with Deluge using the Execute plugin. Read more about *Deluge Integration*.

1.2 Configuration

This section contains addition information and advanced configuration details for EM Media Handler.

If you are a new user, read the *Getting Started* guide first.

1.2.1 Installation Requirements

This is a list of required python packages and 3rd party applications needed by the various parts of EM Media Handler. They are broken down by which configuration section needs them when enabled:

General

- PyYAML (automatically installed)

  ```
  pip install pyyaml
  ```

TV and Movies

- Filebot

Music

- Beets

  ```
  pip install beets
  ```

Audiobooks

- Google API Python Client (automatically installed)

  ```
  pip install google-api-python-client
  ```

- Mutagen (automatically installed)

  ```
  pip install mutagen
  ```

- **ABC (for when `make_chapters` is enabled)** Detailed installation instructions can be found here.

Notifications

- Requests (automatically installed)

  ```
  pip install requests
  ```

Deluge

- Twisted

  ```
  pip install twisted
  ```

- **Deluge** Needs to be installed from source using the directions you can find here.

1.2.2 User Settings

An overview of all available user settings available in the user configuration file. By default, the configuration file is installed here:

```
~/.config/mediahandler/config.yml
```

The configuration file uses YAML formatting, and does not require that every option is present in the file. Sections and options may be left blank or completely removed – the application will use default values in their place.

View an *Example Configuration File*.

General

General mediahandler script functionality options.

Default section and values:

```
General:
    keep_files: no
    keep_if_skips: yes
```

keep_files

Enable or disable mediahandler's removal of the originally downloaded files upon script completion.

Valid options:

- `no` (default)
- `yes`

keep_if_skips

Enable or disable mediahandler's removal of the originally downloaded files in a situation where some of files were skipped during the script's processing.

Valid options:

- `no`
- `yes` (default)

Deluge

Deluge server integration options.

Default section and values:

```
Deluge:
    enabled: no
    host: 127.0.0.1
    port: 58846
    user:
    pass:
```

enabled

Enable or disable mediahandler's ability to automatically remove a torrent from the Deluge UI when the script is executed on torrent completion. Please review the python package and application *Installation Requirements* before enabling.

See *Deluge Integration* for more information on this integration.

Valid options:

- `no` (default)
- `yes`

host

The host IP/address of the running Deluge server.

Default: `127.0.0.1`

port

The port number of the running Deluge server.

Default: `58846`

user

The user running Deluge server (set in the Deluge `auth` file).

pass

The password of the user running Deluge server (set in the Deluge `auth` file).

Logging

Logging output options.

Default section and values:

```
Logging:
    enabled: yes
    level: 30
    log_file:
```

enabled

Enable or disable event logging of the mediahandler script.

Valid options:

- `no`
- `yes` (default)

level

Specify a level threshold for events logged. See this table for possible values.

Default: `30`

log_file

Specify a file path (including file name) to a custom log file destination.

Default: `~/logs/mediahandler.log`

Notifications

Options for push notification via 3rd party services. Multiple services may be used side-by-side.

Default section and values:

```
Notifications:
    enabled: no
    notify_name:
    pushover:
        api_key:
        user_key:
    pushbullet:
        token:
```

enabled

Enable or disable push notifications upon script completion. Please review the python package and application *Installation Requirements* before enabling.

Valid options:

- `no` (default)
- `yes`

notify_name

Specify a name for notifications to use in message titles, e.g. "EM Media Handler: Media Added".

Default: `EM Media Handler`

pushover

To enable Pushover integration, simply set both the `api_key` and `user_key` settings with valid credentials:

```
Notifications:
    enabled: yes
    notify_name: My Custom Name
    pushover:
        api_key: SNAczveGbbyzUmASUljL
        user_key: AkdmliUzQZofvoYVLskG
```

Your `user_key` can be found on your Pushover dashboard.

Your `api_key` is specific to the Pushover application you would like to have the script send the notification through. Click on the application's settings to retrieve the key.

pushbullet

To enable Pushbullet integration, simply set the `token` setting with valid credentials:

```
Notifications:
    enabled: yes
    notify_name: My Custom Name
    pushbullet:
        token: gNJccqGqISParIqHcvRy
```

Your `token` can be found in your Pushbullet account settings.

EM Media Handler does not *yet* support specifying a device or channel to send Pushbullet notifications to.

TV and Movies

TV and Movies both use Filebot and are the only media type modules enabled "out of the box". Their settings are identical in function, which is why they are grouped together in this guide, but they are unique in execution to their respective type.

Default section and values:

```
TV:
    enabled: yes
    folder:
    ignore_subs: yes
    format: "{n}/Season {s}/{n.space('.')}.{'S'+s.pad(2)}E{e.pad(2)}"
    log_file:

Movies:
    enabled: yes
    folder:
    ignore_subs: yes
    format: "{n} ({y})"
    log_file:
```

enabled

Enable or disable processing of media type by mediahandler.

Valid options:

- `no`
- `yes` (default)

folder

Specify a destination folder for added media files.

TV Default: `~/Media/TV`

Movies Default: `~/Media/Movies`

ignore_subs

Tell Filebot whether or not to process subtitle files along with video files or ignore them.

Valid options:

- `no`
- `yes` (default)

format

Specify a Filebot naming format. During processing, it will be appended to the media type's `folder` value to form a complete path. See Filebot's format expressions documentation for more details.

TV Default: `"{n}/Season {s}/{n.space('.')}.{'S'+s.pad(2)}E{e.pad(2)}"`

Movies Default: `"{n} ({y})"`

log_file

Specify a file path (including file name) to a custom log file destination for Filebot to use.

Default: `None` (logging disabled)

Music

The Music media type is integrated with Beets.

Default section and values:

```
Music:
    enabled: no
    log_file:
```

enabled

Enable or disable processing of the music media type by mediahandler. Please review the python package and application *Installation Requirements* before enabling.

Valid options:

- `no` (default)
- `yes`

log_file

Specify a file path (including file name) to a custom log file destination for Beets to use.

Default: `~/logs/beets.log`

Audiobooks

The Audiobook media type makes use of the Google Books API for processing. Additionally, creation of chaptered audiobook files (.m4b) is available via integration with the ABC application for Linux.

EM Media Handler does not *yet* support creation of chaptered audiobook files on OS X.

Default section and values:

```
Audiobooks:
    enabled: no
    folder:
    api_key:
    make_chapters: off
    chapter_length: 8
```

enabled

Enable or disable processing of the audiobooks media type by mediahandler. Please review the python package and application *Installation Requirements* before enabling.

Valid options:

- `no` (default)
- `yes`

folder

Specify a destination folder for added audiobooks.

Audiobooks will be added to the folder in the following format:

```
~/Media/Audiobooks/<author name>/<full book title>/<shorted book title>.<file extension>
```

EM Media Handler does not *yet* support custom renaming formats for Audiobooks.

Default: `~/Media/Audiobooks`

api_key

A valid Google API key. To obtain one, you will need to:

1. Visit the Google API Console.
2. Create a new project (you can keep the default values that Google provides).
3. When your project is created, click on the "Enable an API" button on the Project Dashboard.
4. Scroll to the "Books API" and click on the "Off" button next to it on the right to activate.
5. In the left-hand menu, click on the "Credentials" option under "APIs & auth"
6. Click on the "Create new Key" button under "Public API access".
7. Select "Server key".
8. (Optional) Specify your server's IP for greater security.
9. Copy & paste the generated "API KEY" into the `api_key` setting in your configuration file, e.g.

```
Audiobooks:
    enabled: yes
    folder: /my/path/to/books
    api_key: kKCRCNNsbrfWkohKpxwq
    make_chapters: on
    chapter_length: 8
```

make_chapters

Enable or disable creation of chaptered audiobook files (.m4b) using the ABC application for Linux. Visit the *Installation Requirements* section for information on installation.

EM Media Handler does not *yet* support creation of chaptered audiobook files on OS X.

Values:

- `off` (default)
- `on`

chapter_length

Specify, in *hours*, the maximum length each audiobook file (.m4b) created by ABC should be. For audiobooks that have a running time longer than the specified length, multiple parts will be created, e.g.

```
~/Media/Audiobooks/Donna Tartt/The Goldfinch_ A Novel/The Goldfinch, Part 1.m4b
~/Media/Audiobooks/Donna Tartt/The Goldfinch_ A Novel/The Goldfinch, Part 2.m4b
~/Media/Audiobooks/Donna Tartt/The Goldfinch_ A Novel/The Goldfinch, Part 3.m4b
```

Default: `8` (hours)

1.2.3 Example Configuration File

To learn more about each option, check out the *User Settings* section.

Here is what user configuration file looks like with all available options set:

```
General:
    keep_files: no
    keep_if_skips: yes

Deluge:
    enabled: yes
    host: 192.168.1.6
    port: 58846
    user: deluge
    pass: deluge1

Logging:
    enabled: yes
    level: 30
    log_file: /home/admin/logs/mediahandler.log

Notifications:
    enabled: yes
    notify_name: Home Server
    pushover:
        api_key: snOLInvm7VIBSySbBL9ae1MZmF1xoM
        user_key: utTsCTaOab5FWkoQR4aaCrWtajyWy0
    pushbullet:
        token: xwl2Iex4FRaVVfEMzbvW814G96d3diRY

TV:
    enabled: yes
    folder: /home/admin/media/television
    ignore_subs: yes
    format: "{n}/Season {s}/{n.space('.')}.{'S'+s.pad(2)}E{e.pad(2)}"
    log_file: /home/admin/logs/mediahandler-tv.log

Movies:
    enabled: yes
    folder: /home/admin/media/movies
    ignore_subs: yes
    format: "{n} ({y})"
    log_file: /home/admin/logs/mediahandler-movies.log

Music:
    enabled: yes
    log_file: /home/admin/logs/mediahandler-music.log

Audiobooks:
    enabled: yes
    folder: /home/admin/media/books
```

```
api_key: fbqkyzSfPD0j51gnCeZVNZzBHk576_8PHkSAMHT
make_chapters: on
chapter_length: 8
```

1.2.4 Command-line Interface

An overview of available options and usage of the EM Media Handler command-line interface.

This section is about the `addmedia` script, for information on the `addmedia-deluge` script, visit the *Deluge Integration* section.

Usage

`addmedia` is the primary command-line interface for EM Media Handler.

Basic invocation:

```
addmedia [MEDIA FILES] [OPTIONS...]
```

Use `addmedia --help` at any time to view information on the available options.

Options

These are the options available with the `addmedia` script.

media

REQUIRED. Specify media files to be added. Can be a single file or a folder.

Assumes media has an absolute path structure using this format:

```
/path/to/<media type>/<media>
```

If you are not using this format, you will need to specify a *-t / –type* value.

-t / –type

Force a specific media type for processing. Overrides the detected media type from media.

By default, `addmedia` attempts to detect the media type based on the file path of the media provided. The assumed file path structure is:

```
/path/to/<media type>/<media>
```

Allowed values:

Value	Media Type
1	TV
2	Movies
3	Music
4	Audiobooks

-c / –config

Specify a custom configuration file to use for processing media.

Default: `~/.config/mediahandler/config.yml`

-q / –query

Set a custom query string for audiobooks to pass to the Google Books API.

Useful for fixing "Unable to match" errors, which can occur when a book has a common title and no author name supplied in the file path.

-s / –single

Force beets to import music as in single track mode.

Useful for fixing "items were skipped" errors, especially when a folder contains multiple single tracks instead of an album.

-n / –nopush

Disable push notifications.

This flag overrides the `enabled` setting in the `Notifications` section of the user configuration file, but does not modify it.

Examples

The most basic usage example:

```
addmedia /home/admin/downloads/movies/Finding\ Nemo
```

This would automatically detect the media type `movies` from the folder path name, then move and rename the movie file(s) from within the folder.

If your files are not in a folder named for the correct media type, use the *-t / –type* option to specify what type it is:

```
addmedia /home/admin/downloads/House\ Season\ 1 --type 1
```

This will process the files in the folder as the `1` media type, TV Shows.

Add Audiobooks

The audiobooks module utilizes Google's Books API. It sends a search request to the API based on the file name of the audiobook being added. Most of the time, Google is accurate with just a book name. However, for books with very common-sounding or similar titles, unless the file name contains both the book name and the author's name, we recommend using the *-q / –query* option to specify the exact book information to query Google with.

Good book file name:

```
addmedia /home/admin/downloads/The\ Goldfinch\ Donna\ Tartt --type 4
```

Since the file name has the book title and author, this should match the book information correctly via Google.

Bad book file name and fix:

```
addmedia /home/admin/downloads/Voices --type 4 --query "Voices Arnaldur Indridason"
```

If the `--query` option had not been set for this example, Google would've matched the filename "Voices" to a book called "Voices" by Richard Lortz, not to the book we wanted here, which was "Voices: An Inspector Erlendur Novel" by Arnaldur Indridason.

Fix Music 'Items were skipped' Errors

By default, the Beets application will look for a full album of music to add to your library. It should process single files properly as well. However, for cases where you're trying to add multiple single tracks at once (i.e. a group of songs not from the album) sometimes Beets will throw a matching error or skip the file out of confusion. To fix this issue, use the *-s / –single* flag, which tells Beets to process the files individually, instead of as a group.

For example:

```
addmedia /home/admin/shares/My\ Awesome\ Mixtape --type 3 --single
```

In this example, "My Awesome Mixtape" is a folder containing a bunch of my favorite songs from different artists and albums. The `--single` ensures that beet's processes each file with the correct metadata.

Disable Push Notifications

If push notifications are enabled in your user settings file, the results of any `addmedia` process will create a new push notification. If you need to temporarily suppress these notifications, but don't want to disable them completely, use the *-n / –nopush* option.

Example:

```
addmedia /home/admin/downloads/The\ Fountain --type 2 --nopush
```

Use a Different Configuration File

The configuration file used by EM Media Handler is dependent on the user running the `addmedia` script. By default it looks for `~/.config/mediahandler/config.yml`. If you have a configuration file located elsewhere, or wish to use another user's configuration file, you can specify it with the *-c / –config* option.

Example:

```
addmedia /home/admin/downloads/Orphan\ Black\ Season\ 2 --type 1 --config /home/johnsmith/documents/
```

1.2.5 Deluge Integration

An overview of EM Media Handler's Deluge integration options.

This section is about the `addmedia-deluge` script, for information on the `addmedia` script, visit the *Command-line Interface* section.

Basic Set-up

To utilize the `addmedia-deluge` script, you must have Deluge's Execute plugin installed.

The script is designed to be used with the "Torrent Complete" event.

Note: This is the basic set up and usage of the script. However, using this method, it is assumed that your media is using the EM Media Handler Folder Naming Structure. Read on to the Advanced Set-up section for how to automate this by using an additional Deluge plugin.

To set up the event, all you will need the full path to `addmedia-deluge` script, which can be found by entering the following from the command-line:

```
which addmedia-deluge
```

Which should print something similar to `/usr/local/bin/addmedia-deluge`. Copy and paste this path value into the "Command" text box when adding a new "Torrent Complete" event, save it, and you're done!

Advanced Set-up

The LabelsPlus plugin for Deluge is a great companion to the `addmedia-deluge` script.

The recommended setup:

- Create a label for each media types you intend to use, e.g. "TV", "Movies", etc.
- **Under the "Label Defaults" settings set the following:**
  - Enable the "Enable download settings" option

- Set "Move Completed" to "Label based sub-folder"

You can stop here and manually set each newly added torrent's label, or you can continue on and use the plugin's "Autolabel" settings to create regular expressions that match the various media types and automatically apply labels.

Folder Naming Structure

EM Media Handler's Deluge integration requires that your media download paths follow a certain naming structure so that the correct media type can be detected for processing. The structure is:

```
/path/to/<media type>/<downloaded media>
```

Currently accepted media type values and their case-insensitive variations are:

Type	Allowed Variations
TV	TV, TV Shows, Television
Movies	Movies
Music	Music
Audiobooks	Audiobooks, Books

1.3 For Developers

This section contains information for developers on each of the EM Media Handler python modules. Read on if you wish to extend EM Media Handler's functionality, contribute to the project, or integrate functionality within your own projects.

You can explore the code further on either GitHub or BitBucket, as well as fork your own copy to work with.

1.3.1 mediahandler

Module: mediahandler

Module contains:

- **mediahandler.MHObject** Basic object structure for all module and submodule classes. Contains the MHSettings object, which serves as a simple structure for storing data as attributes.
- Global constants for the module and submodules.

class mediahandler.**MHObject**(**kwargs*)

Bases: object

Base object for the mediahandler module and submodules.

Converts arguments in the form of dict into object attributes for easier data manipulation.

class **MHSettings**(*adict*)

Bases: object

Object which serves as a simple structure for storing data as attributes.

MHObject.**set_settings**(*adict*)

Iteratively converts a dict into MHSettings objects and subsequently into object attributes.

1.3.2 mediahandler.handler

Module: mediahandler.handler

Module contains:

- **mediahandler.handler.MHandler** Main handler object structure which serves as an entry point for the entire module. Contains the core logic for dispatching media to add to their respective submodules for further handling.
- **mediahandler.handler.main()** Wrapper function for handling CLI input.

class mediahandler.handler.**MHandler**(*config*)
Bases: mediahandler.MHObject

Main handler object structure which serves as an entry point for the entire module. Contains the core logic for dispatching media to add to their respective submodules for further handling.

Required argument:

•**config** Full path to valid mediahandler configuration file. A default file available for customization is located here:

```
~/.config/mediahandler/config.yml
```

Public methods:

•**add_media()** Main entry point containing the logic sequence for sending media files to the correct submodule processor.

•**extract_files()** Wrapper function for accessing the mediahandler.util.extract module

class **MHSettings**(*adict*)
Bases: object

Object which serves as a simple structure for storing data as attributes.

MHandler.**add_media**(*media*, ***kwargs*)
Entry point function for adding media via the MHandler object.

Required argument:

•**media** valid path to a file or a folder of media to be added. Assumes structure:

```
/path/to/<media type>/<media>
```

Other valid arguments:

•**type** Int in range [1, 2, 3, 4]. Declare a specific file type. Defaults to a <media type> derived from media path. Valid file types are:

- 1 - TV
- 2 - Movies
- 3 - Music
- 4 - Audiobooks

•**query** String. Set a custom query string for audiobooks. Useful for fixing "Unable to match" errors.

•**single** True/False. Force beets to import music as a single track. Useful for fixing "items were skipped" errors.

•**nopush** True/False. Disable push notifications. Overrides the "enabled" config file setting.

MHandler.**extract_files**(*raw*)
Wrapper function for sending compressed files for extraction via the mediahandler.util.extract module.

Requires the Filebot application for extraction.

MHandler.**set_settings**(*adict*)
Iteratively converts a dict into MHSettings objects and subsequently into object attributes.

mediahandler.handler.**main**(*deluge=False*)
Wrapper function for passing CLI arguments to the MHandler add_media() function for processing.

Optional argument:

- **deluge** True/False. Determines whether basic argument parser or deluge argument parser should be used.

1.3.3 mediahandler.types

Module: mediahandler.types

Module contains:

- **mediahandler.types.MHMediaType** Parent class for all media type submodules. Includes the logic for the video media types (TV & movies).

Media Type Submodules:

- mediahandler.types.audiobooks
- mediahandler.types.movies
- mediahandler.types.music
- mediahandler.types.tv

class mediahandler.types.**MHMediaType**(*settings*, *push*)
Bases: mediahandler.MHObject

Parent class for the media type submodule classes.

Required arguments:

- **settings** Dict or MHSettings object.
- **push** MHPush object.

Public method:

- **mediahandler.types.MHMediaType.add()** Main wrapper function for adding media files. Processes calls to Beets and Filebot.

class MHSettings(*adict*)
Bases: object

Object which serves as a simple structure for storing data as attributes.

MHMediaType.**add**(*file_path*)
Wrapper for Filebot requests.

Sets up Filebot CLI query using object member values.

MHMediaType.**set_settings**(*adict*)
Iteratively converts a dict into MHSettings objects and subsequently into object attributes.

1.3.4 mediahandler.types.audiobooks

Module: mediahandler.types.audiobooks

Module contains:

- **mediahandler.types.audiobooks.MHAudiobook** Child class of MHMediaType for the audiobooks media type.
- **mediahandler.types.audiobooks.get_book_info()** Makes API request to Google Books and returns results.

class mediahandler.types.audiobooks.**MHAudiobook**(*settings*, *push*)
Bases: mediahandler.MHObject

Child class of MHObject for the audiobooks media type.

Required arguments:

- **settings** Dict or MHSettings object.
- **push** MHPush object.

Public method:

- **mediahandler.types.audiobooks.MHAudiobook.add()** Main wrapper function for adding audiobook files. Processes calls to the Google Books API and ABC chaptering tool.

class MHSettings(*adict*)
Bases: object

Object which serves as a simple structure for storing data as attributes.

MHAudiobook.**add**(*raw*)
Main wrapper function for adding audiobook files. Processes calls to the Google Books API and ABC chaptering tool.

Required arguments:

- **raw** Valid path to audiobook files to be processed.

MHAudiobook.**set_book_info**(*query*)
A wrapper function for calling get_book_info().

Converts resulting dict into object members.

MHAudiobook.**set_settings**(*adict*)
Iteratively converts a dict into MHSettings objects and subsequently into object attributes.

mediahandler.types.audiobooks.**get_book_info**(*api_key*, *query*)
Makes API request to Google Books.

Required arguments:

- **api_key** String. A valid Google API public access key.
- **query** String. Search string to submit to Google.

1.3.5 mediahandler.types.movies

Module: mediahandler.types.movies

Module contains:

- **mediahandler.types.movies.MHMovie** Child class of MHMediaType for the movies media type.

class mediahandler.types.movies.**MHMovie**(*settings*, *push*)
Bases: mediahandler.types.MHMediaType

Child class of MHMediaType for the movies media type.

Required arguments:

- **settings** Dict or MHSettings object.
- **push** MHPush object.

Public method:

- **mediahandler.types.movies.MHMovie.add()** inherited from parent MHMediaType.

class MHSettings(*adict*)
Bases: object

Object which serves as a simple structure for storing data as attributes.

MHMovie.**add**(*file_path*)
Wrapper for Filebot requests.

Sets up Filebot CLI query using object member values.

MHMovie.**set_settings**(*adict*)
Iteratively converts a dict into MHSettings objects and subsequently into object attributes.

1.3.6 mediahandler.types.music

Module: mediahandler.types.music

Module contains:

- **mediahandler.types.music.MHMusic** Child class of MHMediaType for the music media type.

class mediahandler.types.music.**MHMusic**(*settings*, *push*)
Bases: mediahandler.types.MHMediaType

Child class of MHMediaType for the music media type.

Required arguments:

- **settings** Dict or MHSettings object.
- **push** MHPush object.

Public method:

- **mediahandler.types.music.MHMusic.add()** inherited from parent MHMediaType.

class MHSettings(*adict*)
Bases: object

Object which serves as a simple structure for storing data as attributes.

MHMusic.**add**(*file_path*)
Overrides the MHMediaType object to process Beets requests.

Sets up Beets CLI query using object member values.

MHMusic.**set_settings**(*adict*)
Iteratively converts a dict into MHSettings objects and subsequently into object attributes.

1.3.7 mediahandler.types.tv

Module: mediahandler.types.tv

Module contains:

- **mediahandler.types.tv.MHTv** Child class of MHMediaType for the TV media type.

class mediahandler.types.tv.**MHTv**(*settings*, *push*)
Bases: mediahandler.types.MHMediaType

Child class of MHMediaType for the TV media type.

Required arguments:

- **settings** Dict or MHSettings object.
- **push** MHPush object.

Public method:

- **mediahandler.types.tv.MHTv.add()** inherited from parent MHMediaType.

class MHSettings(*adict*)
Bases: object

Object which serves as a simple structure for storing data as attributes.

MHTv.**add**(*file_path*)
Wrapper for Filebot requests.

Sets up Filebot CLI query using object member values.

MHTv.**set_settings**(*adict*)
Iteratively converts a dict into MHSettings objects and subsequently into object attributes.

1.3.8 mediahandler.util

Module: mediahandler.util

This module contains submodules which provide utility functions to the main mediahandler object.

Submodules:

- **mediahandler.util.args** Retrieves and parses argument input from the CLI.
- **mediahandler.util.config** Retrieves and parses user settings from the configuration file provided.
- **mediahandler.util.extract** Uses Filebot to extract compressed files for processing.
- **mediahandler.util.notify** Sends push notifications out via 3rd party services.
- **mediahandler.util.torrent** Removes torrents from Deluge upon completion.

1.3.9 mediahandler.util.args

Module: mediahandler.util.args

Module contains:

- **mediahandler.util.args.MHParser**

  Custom argparse.ArgumentParser child object.

  - **mediahandler.util.args.get_parser()** Retrieves full parser object for CLI args.

  - **mediahandler.util.args.get_deluge_parser()** Retrieves abbreviated parser for Deluge args.

- **Custom argparse.Action validation objects**
  - **mediahandler.util.args.MHMediaAction** For parsing media path structure.
  - **mediahandler.util.args.MHFilesAction** For checking if files or folders exist.
  - **mediahandler.util.args.MHTypeAction** For validating media type.
- **Argument validation wrapper functions:**
  - **mediahandler.util.args.get_arguments()** Wrapper for the 'addmedia' CLI.
  - **mediahandler.util.args.get_deluge_arguments()** Wrapper for the 'addmedia-deluge' CLI.
  - **mediahandler.util.args.get_add_media_args()** Wrapper for the mediahandler.handler.add_media() function.

class mediahandler.util.args.**MHFilesAction**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

Bases: argparse.Action

Custom files and folders validation action for argparse.

A child object of argparse.Action().

class mediahandler.util.args.**MHMediaAction**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

Bases: argparse.Action

Custom media validation action for argparse.

A child object of argparse.Action().

class mediahandler.util.args.**MHParser**(*prog=None*, *usage=None*, *description=None*, *epilog=None*, *version=None*, *parents=[]*, *formatter_class=<class 'argparse.HelpFormatter'>*, *prefix_chars='-'*, *fromfile_prefix_chars=None*, *argument_default=None*, *conflict_handler='error'*, *add_help=True*)

Bases: argparse.ArgumentParser

A child object of argparse.ArgumentParser().

Extends the following methods:

- parse_known_args()
- error()
- print_help()

error(*message*)

Displays a simple help message via stdout before error messages.

Overrides the argparse.ArgumentParser() method.

parse_known_args(*args=None*, *namespace=None*)

Saves the array of args as 'entered' to the argparse.Namespace() object for use in the MHMediaAction() object. Removes the added values before returning the validated args.

Extends the argparse.ArgumentParser() method.

print_help(*files=None*)
Makes the printed help message look nicer by adding padding before and after the text and by adding the program's title and version information in the header.

Extends the argparse.ArgumentParser() method.

class mediahandler.util.args.**MHTypeAction**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)
Bases: argparse.Action

Custom media type validation action for argparse.

A child object of argparse.Action().

mediahandler.util.args.**get_add_media_args**(*media*, ***kwargs*)
Takes arguments passed in from the mediahandler.handler.add_media() function, formats them into an array, and sends them to get_parser() for validation.

Returns a dict of validated arguments from the MHParser object.

mediahandler.util.args.**get_arguments**(*deluge=False*)
Retrieves CLI arguments from the 'addmedia' script and uses get_parser() to validate them.

Returns the full file path to the config file in use and a dict of validated arguments from the MHParser object.

mediahandler.util.args.**get_deluge_arguments**()
Retrieves CLI arguments from the 'addmedia-deluge' script and uses get_deluge_parser() to validate them.

Returns the full file path to the config file in use and a dict of validated arguments from the MHParser object. Also removes the torrent from Deluge, if the user's settings allow it.

mediahandler.util.args.**get_deluge_parser**()
Returns the custom MHParser object for mediahandler usage.

Sets up the MHParser object details and adds all of the arguments used by the mediahandler CLI 'addmedia-deluge' script.

mediahandler.util.args.**get_parser**()
Returns the custom MHParser object for mediahandler usage.

Sets up the MHParser object details and adds all of the arguments used by the mediahandler CLI 'addmedia' script.

1.3.10 mediahandler.util.config

Module: mediahandler.util.config

Module contains:

- **mediahandler.util.config.make_config()** Generates a default yaml configuration file.
- **mediahandler.util.config.parse_config()** Parses a yaml configuration file and returns a dict of the settings.

mediahandler.util.config.**make_config**(*new_file=None*)
Generates default yaml mediahandler configuration file.

Optional argument:

- **new_file** Path to a yaml mediahandler configuration file. Will verify that the file exists and is readable.

mediahandler.util.config.**parse_config**(*file_path*)
Reads and parses a yaml mediahandler configuration file.

Optional argument:

- **file_path** Path to a valid yaml mediahandler configuration file.

Uses settings.yml validation structure to build missing and default values. Sends values to the correct _get_valid_<type>() function for validation.

1.3.11 mediahandler.util.extract

Module: mediahandler.util.extract

Module contains:

- **mediahandler.util.extract.get_files()** extracts compressed files via Filebot.

mediahandler.util.extract.**get_files**(*filebot*, *file_name*)
Extracts compressed files via Filebot.

Required arguments:

- **filebot** Path to valid Filebot application script.
- **file_name** Path to valid compressed file for extraction.

1.3.12 mediahandler.util.notify

Module: mediahandler.util.notify

Module contains:

- **mediahandler.util.notify.MHPush** An object which contains the all the logic for sending push notifications and raising errors produced by mediahandler.

class mediahandler.util.notify.**MHPush**(*settings*, *disable=False*)
Bases: mediahandler.MHObject

An object which contains the all the logic for sending push notifications and raising errors produced by mediahandler.

Arguments:

- **settings** Required. Dict or MHSettings object.
- **disable** True/False.

Public methods:

- **mediahandler.util.notify.MHPush.send_message()** Wrapper function for sending push notification messages via various 3rd party services.
- **mediahandler.util.notify.MHPush.success()** A wrapper for send_message() which sends a success message and returns message content.
- **mediahandler.util.notify.MHPush.failure()** A wrapper for send_message() which sends a failure message and raises a SystemExit.

class MHSettings(*adict*)
Bases: object

Object which serves as a simple structure for storing data as attributes.

MHPush.**failure**(*error_details*)
Builds and sends a failure notification.

Required argument:

- **error_details** String. A message detailing the error that was reported during the add_media() sequence.

MHPush.**send_message**(*conn_msg*, *msg_title=None*)
Wrapper for sending push notifications via 3rd party services.

The function will exit if the disable flag is set.

MHPush.**set_settings**(*adict*)
Iteratively converts a dict into MHSettings objects and subsequently into object attributes.

MHPush.**success**(*file_array*, *skipped=None*)
Builds and sends a success notification.

Arguments:

- **file_array** Required. An array of the files added via the mediahandler.handler.add_media() function.
- **skipped** Defaults to None. An array of any files that were skipped during the add_media() sequence.

1.3.13 mediahandler.util.torrent

Module: mediahandler.util.torrent

Module contains:

- **mediahandler.util.torrent.remove_deluge_torrent()** Removes a torrent from Deluge.

mediahandler.util.torrent.**remove_deluge_torrent**(*settings*, *torrent_hash*)
Removes a torrent from Deluge.

Required arguments:

- **settings** Dict or MHSettings object for Deluge info.
- **torrent_hash** Valid hash of active torrent to be removed.

This is a Twisted Deferred object which hooks into the Deluge UI client. For more information, visit: http://dev.deluge-torrent.org/wiki/Development/UiClient1.2

m

A

E

F

G

M

P

R

S