MediaList

for version 0.4, 2022-04-06

mio (stigma@disroot.org)

This manual is for MediaList (version 0.4, 2022-04-06), a library for interacting with medialist files.

Table of Contents

1 Overview

'medialist' is a D package for interacting with mTSV (m? Tab Separated Values) files. These files are rather similar to normal *TSV* files, but differ in that they support comments. Basing the "list"s off of the *TSV* format means that they're just plain-text, allowing people to still edit them manually with a text editor.

The package is designed to help create and maintain *mTSV* files, with a focus on reading and watching media. It works by sending various "commands" to the list, along with some arguments. See [MLCommand], page 5, for an overview of the different "commands".

2 mTSV

The mTSV file format is based off the TSV (Tab Separated Values) file format. The complete file represents a data table where each value is separated by the tab character (`0x09`, *`HORIZONTAL TAB`*). The rows are delimitated by a new line (`0x0A`, *`LINE FEED`*). So far this is the same as a regular *TSV* file. The only difference with a *mTSV* file is with the addition of "comments". If a line starts with an octothorpe (`0x23`, `#`), then it is interpreted as a comment and the line can be ignored.

Some programs (e.g. the medialist suite of programs (`https://yume-neru.neocities.org/p/medialist.html`)) use these comments as a way of providing per-file configuration.

3 Constants

There are only a couple of constants present in MediaList.

`enum string MediaListVersion`
: This string represents the current version of MediaList, it may also contains a "-dev" suffix to indicate that you're running a development version which may have some known bugs. The format is Major.Minor.Patch.

  A "Major" version change represents the removal or modification of an existing API. The "Patch" version is used only for compile-time fixes. The "Minor" version represents all other changes.

`enum int[3] MediaListVersionA`
: This static array represents the version of MediaList you're compiling against. Unlike the string version, this doesn't have a method for determining a development version. The format is still Major.Minor.Patch.

> **Note**: MediaList hasn't reached a "1.0.0" version yet, each new release is only incrementing the "Minor" version. These "pre-Major" versions can still contain modifications to (or the removal of) APIs.

4 Data Types

An overview of all the public data types that this package provides, their fields (if any), and what their purpose is.

4.1 MediaList

The first (and most important) structure is the '`MediaList`' structure. This structure is passed to all functions that are defined in this package.

It is recommended to create an instance by using the `ml_open_list` function.

It has the following members:

'`immutable string filePath`'
: The absolute file path to the *TSV* file. Upon creating a '`MediaList`' structure, a check is performed to see if this file exists. If it doesn't exist, it is created.

'`immutable string listName`'
: The name of the "list". This is determined by extracting the file name from the '`filePath`'. For example, if the '`filePath`' is `/home/user/lists/mylist.tsv`, then the '`listName`' would be '`mylist`'.

'`bool isOpen`'
: A check to see if the file at '`filePath`' is open. All of the functions which edit the file will check this before doing anything. By providing public access to this, I hope people can make their own functions to suit there needs without breaking any of the ones provided in this package.

> **NOTE:** The future of this property is being evaluated. It requires people using this library to remember that it exists, and performing checks on its value. It is probable that this will be removed in a future version.

4.2 MediaListHeader

Contains information about a particular "header" (i.e. column) in the *mTSV* file. Editing these values *does not change their values in the list.*

'`string tsvName`'
: The literal column name as written in the *TSV* file.

'`string humanFriendlyName`'
: The "human friendly name" as determined by the comments (see Chapter 2 [mTSV], page 2). If this value is set, then it should be used when displaying the header to people. The default value is `null`.

'`size_t column`'
: The column number that this header is for. Useful if you want to read from the file directly.

4.3 MediaListItem

A structure that is returned from all of the "fetch" functions. Editing these values *does not change their values in the list.*

'`string title`'
: The title of this item in the list.

'`string progress`'
: The current progress of this item in the list. By default this is '`-/-`'.

'`string status`'
: The current status of this item in the list. By default this is '`UNKNOWN`'.

'`string startDate`'
: The date that this item was "started" on. The format for this is '`YYYY-MM-DD`'. This value is automatically updated when the '`status`' changes from '`PLAN-TO-READ`' or '`PLAN-TO-WATCH`' to '`READING`' or '`WATCHING`' respectively.

'`string endDate`'
: The date that this item was "completed" on. The format for this is '`YYYY-MM-DD`'. This value is automatically updated when the '`status`' changes from '`READING`' or '`WATCHING`' to '`COMPLETE`'.

'`string lastUpdated`'
: The date that this item was last updated on. The format for this is '`YYYY-MM-DD`'. This is automatically updated whenever this item is updated.

'`bool valid`'
: A check to see if this item was initialized correctly upon being fetched. If it is `false`, then it failed and the values shouldn't be trusted.

> **NOTE**: The future of this property is undecided. It requires that programmers remeber it exists, and to make use of it. With MediaList 0.4, most functions throw an exception (See [MLException], page 7) if there is an error, or return an error type (See [MLError], page 6) when an error occurs a "nothrow" variant of a function has been called.

'`string tsvRepresentation()`'
: This member function will convert a '`MediaListItem`' structure in to a TSV representation. This does **not** add a trailing newline character. The order follows the MediaList default order:

  ```
  title, progress, status, start_date, end_date, last_updated
  ```

4.4 MLCommand

This enumeration type represents the type of command when sending a list of arguments to `ml_send_command` when updating a '`MediaList`'.

`add`

By providing the "add" command to `ml_send_command`, you are indicating that you want to append a new item to the list. The format of the arguments (*string[] args*) is as follows:

```
args[0] = The new item's Title
args[1] = The new item's Progress
args[2] = The new item's Status
```

Aside from the Title, any of the arguments can be `null` to use the defaults ("-/-" for Progress and "UNKNOWN" for Status).

`delete_`

By sending the "delete_" command to a list, you're saying that each item ID in the provided arguments should be removed from the list. For example:

```
ml_send_command(list, MLCommand.delete_, ["1", "10", "3"]);
```

The order of the IDs does not matter, this will remove the items with IDs 1, 3, and 10.

NOTE: Previously you could send an empty array as an argument to delete the list file. This behaviour is deprecated and will be removed in a future version.

`update`

This will allow you to update items in the list. The first argument is the item ID you want to update. The remaining arguments are the values that you want to change. The format is as follows:

```
args[0] = Item ID (e.g. "1").
args[n] = field::value
```

The double colons ('`::`') acts as a separator.

The possible values for "field" are:

- title
- progress
- status
- start_date
- end_date

The "value" can contain spaces.

4.5 MLError

`success`

This value indicates that the function worked without any errors. You can proceed as normal. The value of '`success`' is 0.

`invalidArgs`

This value indicates that the arguments provided weren't sufficient. For example, sending an empty array with the `MLCommand.add` command. The value of '`invalidArgs`' is 1.

`fileAlreadyOpen`
: This error is returned when a `MediaList` structure is in an "open" state. All the functions provided in this package check the file isn't open before reading/writing. Should you choose to extend this package by writing your own functions, this can prevent accidental read/write errors. The value of '`fileAlreadyOpen`' is 2.

`fileReadError`
: An error was encountered when reading the list file. The value for '`fileReadError`' is 3.

`itemNotFound`
: A requested item (by `ml_fetch_item` or `ml_fetch_items`) could not be found. The value of '`itemNotFound`' is 4.

`permissionError`
: When trying to create or open the file for a `MediaList`, we encountered a permission error and couldn't open the file. The value for '`permissionError`' is 5.

`unknownCommand`
: An unknown command was provided to the *command* parameter for the `ml_send_command` function. The value of '`unknownCommand`' is 6.

`unspecifiedError`
: An unknown error occurred. This is used as a fallback for operating system and Phobos exceptions. The value of '`unspecifiedError`' is always equal to `MLError.max`.

4.6 MLException

A custom MediaList Exception class which is used for error reporting in the functions which throw (rather than the `nothrow` variants).

`MLError error()`
: Returns the underlying `MLError` value.

`string msg()`
: Return a string describing the reason for the exception.

`this(string msg)`
: Construct a new `MLException` by providing only an error message.

`this(MLError error)`
: Construct a new `MLException` by providing a `MLError` code. The error message is determined by this value.

`this(string msg, MLError error)`
: Construct a new `MLException` by providing a custom message and error code.

5 Functions

`MediaList* ml_open_list(string filePath)`
`MediaList* ml_open_list(string filePath, out MLError error) nothrow`

This function will allocate the memory for a `MediaList` structure, returning `null` if the allocation fails.

If the underlying file for the list doesn't exist (i.e. a new list is being created), then the file will be created with the default headers.

The nothrow version will return provide a `MLError` in the event of an error when creating the new file, otherwise it will be equal to `MLError.success`.

`void ml_free_list(MediaList* list) nothrow`

This function will free the allocated memory for a `MediaList` structure created by `ml_open_list`.

`MediaListHeader[] ml_fetch_headers(MediaList* list)`
`MediaListHeader[] ml_fetch_headers(MediaList* list, out MLError error) nothrow`

This function will fetch all of the headers from the list's mTSV file. An optional `error` parameter is provided for error handling.

`MediaListItem ml_fetch_item(MediaList* list, size_t id)`
`MediaListItem ml_fetch_item(MediaList* list, size_t id, out MLError error) nothrow`

Fetch a single item from the list.

`MediaListItem[] ml_fetch_items(MediaList* list, size_t[] ids...)`
`MediaListItem[] ml_fetch_items(MediaList* list, out MLError err, size_t[] ids...) nothrow`

Fetch multiple items from a list.

`MediaListItem[] ml_fetch_all(MediaList* list)`
`MediaListItem[] ml_fetch_all(MediaList* list, out MLError error) nothrow`

Fetch and return all of the items in a list.

`void ml_send_command(MediaList* list, MLCommand command, string[] args)`
`void ml_send_command(MediaList* list, MLCommand command, string[] args, out MLError error) nothrow`

Send a command to the list. This is how all of the editing of the list is done. The format of *args* depends on what the *command* is.

6 Example

A simple example that demonstrates most aspects of the library.

```
// I (mio, <stigma@disroot.org>) have released this example to the
// public domain. For more information, visit
// <http://creativecommons.org/publicdomain/zero/1.0/>.

import std.stdio : stderr, writeln, writefln;
import std.file : remove;

import medialist;

void print_error(MLError err)
{
  stderr.writefln("Error: MLError.%s", err);
}

int main(string[] args)
{
  if (2 > args.length)
    {
      stderr.writefln("usage: %s <path_to_list.tsv>", args[0]);
      return 1;
    }

  /* Open a new list for editing, creating the file if needed */
  MediaList* list = ml_open_list(args[1]);

  /* Make sure the memory is returned when the program exits. */
  scope(exit) ml_free_list(list);

  /* Since we don't want to keep the file for this example. */
  scope(exit) remove(list.filePath);

  /* Error handling */
  MLError err;

  /*
   * Add some items to the list.
   *
   * The first has default progress (-/-) and status (UNKNOWN).
   * The second has default status (UNKNOWN).
   * The third has default progress (-/-).
   */
  ml_send_command(list, MLCommand.add, ["Item 1"], err);
  if (MLError.success != err) print_error(err);
```

```
ml_send_command(list, MLCommand.add, ["Item 2", "10/100"], err);
if (MLError.success != err) print_error(err);

ml_send_command(list, MLCommand.add,
                ["Item 3", null, "PLAN-TO-WATCH"], err);
if (MLError.success != err) print_error(err);

ml_send_command(list, MLCommand.add, ["Item 4", "10/10", "COMPLETE"],
                err);


/*
 * Update some items.
 *
 * We update the progress and status of item 1.
 * We update the start date and end date of item 4.
 * We update the title of item 3.
 */
ml_send_command(list, MLCommand.update,
                ["1", "progress::0/100", "status::PLAN-TO-READ"], err);
if (MLError.success != err) print_error(err);

ml_send_command(list, MLCommand.update,
                ["4","start_date::2021-11-25","end_date::2022-02-13"],
                err);
if (MLError.success != err) print_error(err);

ml_send_command(list, MLCommand.update,
                ["3", "title::The Third Item"], err);
if (MLError.success != err) print_error(err);

/* Fetch an item and write the details out. */
MediaListItem item4 = ml_fetch_item(list, 4, err);

if (MLError.success == err && true == item4.valid)
  {
    writeln("Details about item4:");
    writefln("Title:       %s", item4.title);
    writefln("Progress:    %s", item4.progress);
    writefln("Status:      %s", item4.status);
    writefln("Start Date:  %s", item4.startDate);
    writefln("End Date:    %s", item4.endDate);
    writefln("Last Update: %s", item4.lastUpdated);
  }
else
  {
    stderr.writefln("Failed to fetch item4: MLError.%s (Valid? %s)",
```

```
                    err, item4.valid ? "true" : "false");
    }

  /*
   * Delete some items.
   */
  ml_send_command(list, MLCommand.delete_, ["3", "2"], err);
  if (MLError.success != err) print_error(err);

  /* Done ! */
  return 0;
}
```

Index