AEMT -**A**tari **E**ight-bit **M**ulti-**T**ool

This is a command-line tool, intended to provide some "quality of life" improvements to managing large game/program libraries, on USB Media, for the "Atari THE400 Mini" (with various functions having utility for other Atari 8-bit concerns).

Current functionality includes:

- Moving and splitting files, in bulk, to organized folder structures - with the ability to control how the organization is done and to limit the number of files per folder (to avoid exceeding "THE400 Mini's" 255 files-per-folder limit).

- Creating, applying, and updating ".cfg" files automatically, **without** having to set them individually, one game at a time, from within "THE400 Mini's" file browser.

- Identify and validate CARTRIDGE files (".car" and ".c01" to ".c70"), including decoding the header, showing the stored ROM data checksum and computing the actual checksum of the contained ROM data.

Note: Only the ".cfg" file functionality is specific to "THE400 Mini". Other functions can be used with *any* Atari 8-bit emulators/systems.

For example, applying a ".cfg" file to **all** the games/programs in a complex, nested, folder-structure is as simple as:

```
aemt.py config apply my_defaults.cfg "/emulation/atari/games"
```

Or to split all your games from one set of folders, into another that contains no more than 250 files each (`-m 250`), and which are organized by the leading characters for the first, and last, game in each folder, and grouped to keep like-prefixes together (`-g`):

```
aemt.py split max -m 250 -g -c "/atari games/" "/new/games/folders/"
```

Basic Usage

Run the AEMT tool, and see basic help[1]:

```
aemt.py --help
```

This will yield the display, below, which shows the three main commands available:

- **cart** – Idenfitifes and validates Atari 8-bit cartridges
- **config –** Creates and updates .CFG files for THE400 Mini USB Media games
- **split –** Moves files to organized folder structures.

```
Usage: aemt.py [OPTIONS] COMMAND [ARGS]...

  [A]tari [E]ight-bit [M]ulti-[T]ool

      Moves files to organized folder structures, with an optionally
      limited number of files per folder.

      Creates, applies and updates .cfg files for Atari THE400 Mini
      games on USB media.

      Identifies and verifies Atari 8-bit cartridge images.

Options:
  --version  Show the version and exit.
  --help     Show this message and exit.

Commands:
  cart    Identifies and validates Atari 8-bit cartridges.
  config  Creates and updates .CFG files for THE400 Mini USB Media games.
  split   Moves files to organized folder structures.
```

To get help for specific commands you can add the command of interest[2]:

```
aemt.py split --help
```

[1] If you've setup the scripts to be executable (see "Installation Etc."), then you can run them as shown, otherwise you need to prefix each command with "`python `"; e.g., "`python aemt.py --help`".
[2] All colors shown in commands and examples are for emphasis only, and do not appear in the actual program input or output.

```
Usage: aemt.py split [OPTIONS] COMMAND [ARGS]...

  Moves files to organized folder structures.

Options:
  --version  Show the version and exit.
  --help     Show this message and exit.

Commands:
  alpha  Splits source files into smaller, alphabetic/numeric folders.
  band   Splits source files into smaller, "bands" (e.g., 0-9, A-E etc.).
  max    Splits source files into folders, with a max # of files each.
```

And from there you can ask for help on the specific sub-commands:

```
aemt.py split max --help
```

Which will give you the details for all options and arguments needed for that operation:

```
Usage: aemt.py split max [OPTIONS] [SOURCE_PATH] [DEST_PATH]

  Splits source files into folders, with a max # of files each. Optionally,
  tries to group files with like-prefixes together (may result in fewer than
  the max # of files per folder).

Options:
  -c, --copy                Copies files to new partitions (else does nothing)
  -d, --delete              Deletes original files after copying them (move)
  -g, --group               Groups files by like prefixes
  -m, --max_files INTEGER   Maximum # of files per partition  [default: 250]
  -v, --verbosity [0|1|2]   Status/progress reporting verbosity  [default: 1]
  --help                    Show this message and exit.
```

This same, hierarchical, command – sub-command – options approach works for all commands and functions of this tool.

“Config” Command & Configuration Files (“.cfg”):

AEMT has three basic functions for dealing with “.cfg” files for “THE400 Mini”, all of which work in concert to provide flexibility and power.

Creating/editing .cfg files is handled separately from **applying** or **updating** them.

This means you can **create new** .cfg files with *this* tool, *or* create them with “THE400 Mini” itself, or use existing .CFG files, and have them applied to one, or ALL, of your games/programs with a single command.

You can also create “update” configurations, that bulk-change one or more values in **existing** configuration files, *without* altering their other settings.

For example, if you decide you want to make ALL your files use new display settings, but don’t want to change any other settings, you can create an “update.cfg” file that contains JUST the values to be changed (or added),

Create

The “`create`” command produces new, complete, .CFG files using input on the command line. You can specify the machine/model, which video standard you want (NTSC or PAL), whether it should have BASIC enabled, display start/height/width and so on:

```
Usage: aemt.py config create [OPTIONS] [FILENAME]

  Creates .cfg files for Atari THE400 Mini.

Options:
  -m, --model [400|800|XL|XE|5200]
                                  Atari 8-bit Model to configure for (where
                                  XL=800XL and XE=130XE)  [default: XE]
  -b, --basic                     Enable BASIC
  -vs, --video_std [NTSC|PAL]     Video Standard (NTSC=60Hz, PAL=50Hz)
                                  [default: NTSC]
  -p, --force_pal                 Forces PAL operation (60Hz refresh)
  -a, --artefact [0|1|2|3]        Artefacting mode  [default: 0]
  -s, --start INTEGER             Display Start scanline (lower values start
                                  higher on screen)  [default: 20]
  -h, --height INTEGER            Display Height (scanlines)  [default: 200]
  -w, --width INTEGER             Display Width (pixels)  [default: 320]
  --help                          Show this message and exit.
```

Note: You cannot, currently, use the `create` command line to *customize* controller mappings; however, both the "apply" and "update" functions will respect any manual edits you make to these. You can also use the normal "THE400 Mini" configuration interface to create a suitable .CFG file, and then use IT with the "apply" and "update" functions.

For example, to create a new .CFG file, which specifies the 130XE as the machine model, without BASIC enabled, with the display settings of start scanline of 14, a height of 206 pixels and a width of 320, for PAL you would enter, and save it as "`myfile.cfg`":

```
aemt.py config create -m XE -vs PAL -s 14 -h 206 -w 320 myfile.cfg
```

In this example, it is not necessary to *specify* options for BASIC, as that is disabled by default. If you wanted to enable it, you'd add "`-b`" before the filename.

Apply

The "`apply`" command will let you take **any** ".cfg" file for "THE400 Mini" (such as the one created above) and apply it to one, or more, other games/programs:

```
Usage: aemt.py config apply [OPTIONS] CONFIG_FILE DEST_PATH

  Applies specified .cfg file to THE400 Mini USB Media games.

Options:
  -o, --overwrite          Overwrite existing .cfg files
  -r, --recurse            Recursively process all files in target directory
  -v, --verbosity [0|1|2]  Status/progress reporting verbosity  [default: 1]
  --help                   Show this message and exit.
```

This means you can create **new** .CFG files with this tool, create them with "THE400 Mini" itself, or use existing .CFG files, and have them applied to one, or **all**, of your games/programs with a single command.

If you already had the perfect configuration for "Galaxian" and you wanted to copy it to **all** your other games, but without touching games that already have a .cfg file, you would run:

```
aemt.py config apply -r /games/Galaxian.cfg /games/
```

Unless you add the "`-o`" option, the "`apply`" command will **not** touch any existing .cfg files!

Update

The "`update`" command takes the content of a partial .cfg file (e.g., just the settings for the screen size), and updates existing .cfg files to match – **without** changing *other* settings:

```
Usage: aemt.py config update [OPTIONS] UPDATE_FILE DEST_PATH

  Updates .cfg files with settings from specified update file.

Options:
  -r, --recurse            Recursively process all files in target directory
  -v, --verbosity [0|1|2]  Status/progress reporting verbosity  [default: 1]
  --help                   Show this message and exit.
```

Warning: Unlike "`apply`", the "`update`" command **will** modify **every** .cfg file it encounters (as that's the entire point of the command), so be sure about what you specify for "`DEST_PATH`" (which can be a single file or folder) and whether you mean to recursively process sub-folders.

To make updates, you need to create a "partial" .cfg file, containing *just the values you want to update*, for example a file called "`big_display.cfg`":

```
display_start_line = 10;
display_height = 220;
display_width = 320;
```

And then run the following command to update **just those values** for all the games in the folder "`/games/A-E/`":

```
aemt.py config update big_display.cfg /games/A-E/
```

“Split” Command & File/Folder Splitting:

AEMT has three ways of splitting up folders/files to address organization and keeping the number of files per folder under the 255 file limit for “THE400 Mini”:

```
Usage: aemt.py split [OPTIONS] COMMAND [ARGS]...

  Moves files to organized folder structures.

Options:
  --version  Show the version and exit.
  --help     Show this message and exit.

Commands:
  alpha  Splits source files into smaller, alphabetic/numeric folders.
  band   Splits source files into smaller, "bands" (e.g., 0-9, A-E etc.).
  max    Splits source files into folders, with a max # of files each.
```

The sub-commands work as follows (see “Common Usage Examples:” for examples):

- **alpha** – Takes all the files from all the folders in the specified “`source_path`” and re-organizes them into folders named for individual letters and numbers.
    - So, all files beginning with “A” go in a folder called “A”, files beginning with “B” go into a folder called “B” and so on.
    - It will **only** create folders if there are files that belong in them, so if you have no files beginning with “Z” you won’t see a “Z” folder.
    - There is **no limit** to the number of files each folder can contain, so with large collections, this **can** result in a folder having more files in it than “THE400 Mini” will display.
- **band** – Takes all the files from all the folders in the specified “`source_path`” and re-organizes them into folders that span multiple, initial, character values.
    - If you specify bands: “0-9,A-M,N-Z” you’ll see three folders created – “0-9”, “A-M” and “N-Z”, and all the files beginning will be copies to those folders based on the first character in their names.
    - There is **no limit** to the number of files each folder can contain, so with large collections, this **can** result in a folder having more files in it than “THE400 Mini” will display.

- **max** – Takes all the files from all the folders in the specified "`source_path`" and re-organizes them into the *smallest number of folders possible* where each folder contains **no more** than a **specified** number of files.

    - If you specify 250 files per folder that is the **maximum** number of files any folder will contain (it can be less if using *grouping*, or for the last folder).

    - Folders are named for the unique starting/ending file names, e.g. "SA-SP" or "Thev-U" etc.

    - The "`-g`" option attempts to keep files with similar prefixes together, which results in shorter folder names (as there are fewer overlaps), but also will result in some folders having **less** than the maximum specified number of files.

Note: All sub-commands for "`split`" require you specify the "`-c`" option in order for them to **actually copy files**. Otherwise ,they simply preview the list of files and, if you have the "`-v 2`" option set, display what they're going to do specifically. This is so you can verify the result before making copies of files in a new structure.

Note: The various "split" commands do **NOT delete** *any* files. You can do this yourself using your O.S.' UI or command line easily, so it is safer not to include here.

“Cart” Command & Cartridge Files:

AEMT currently has two commands for dealing with cartridges:

```
Usage: aemt.py cart [OPTIONS] COMMAND [ARGS]...

  Identifies and validates Atari 8-bit cartridges.

Options:
  --version  Show the version and exit.
  --help     Show this message and exit.

Commands:
  id          Identifies the cartridge type and verifies header and data.
  list_types  Lists known cartridge types and specifications.
```

These work as follows:

- **list_types** – displays a list of all known cartridge types (refer to this list), sizes, machine compatibility and descritpions.
- **id –** identifies a cartridge, or list of cartridges, and displays their:
    - **Signature** – which for a valid cartridge image will be: “`CART`”
    - **Type –** The type of cartridge the image represents (refer to this list).
    - **Checksum** – this is the checksum, in hexadecimal, for the ROM data as *stored in the cartridge header*.
    - **Actual Checksum** – this is the checksum, in hexadecimal computed from the *current ROM data* in the cartridge image (**not** the header).
    - **True/False** – Whether the cartridge appears to be a valid image.
    - **Description** – Text describing the type and nature of the cartridge (again, refer to this list).

  Additional options exist to allow output in .CSV format, if processing multiple files, with or without a header row, and the ability to include additional data:

```
Usage: aemt.py cart id [OPTIONS] SOURCE_PATH

  Identifies the cartridge type and verifies header and data.

    SOURCE_PATH may be a directory or a file; if a directory *only* .car files
    will be processed.  The -r/--recurse option will include subdirectories.

Options:
  -c, --csv      Output in CSV format
  -h, --header   Output a header if in CSV format
  -r, --recurse  Process directories recursively for .car files
  -v, --verbose  Verbose output
  --help         Show this message and exit.
```

Common Usage Examples:

Split - Examples

Create a reorganized copy of game files/folders, with no more than 50 games per folder, and grouping games by the first character of their file names (where possible):

```
aemt.py split max -c -g -m 50 “current_games_folder/” “new_base_folder/”
```

Organize game files/folders into specific bands (0-9, A-J, K-S and T-Z):

```
aemt.py split band -c “0-9,A-J,K-S,T-Z” “current_games_folder/” “new_base_folder/”
```

Cart - Examples

Identify a cartridge:

```
aemt.py cart id “/games/O-R/Pac-Man.car”
```

Create a list of cartridge IDs for a folder, in CSV format, with a header and expanded info:

```
aemt.py cart id -c -h -v “/games/”
```

Config – Examples

Apply an existing “personal_default.cfg” to all games in the folder “A-E”, even if they already have a .CFG file:

```
aemt.py config apply -o personal_default.cfg /games/A-E
```

Apply the “Missile Command.cfg”, created by “THE400 Mini” only to games that DO NOT have a .CFG file:

```
aemt.py config apply “Missile Command.cfg” /games/
```

Create a new .CFG file, which specifies the 130XE as the machine model, with BASIC enabled, with the display settings of start scanline of 14, a height of 206 pixels and a width of 320, for NTSC you would enter, and save it as "`my130XEconfig.cfg`":

```
aemt.py config create -m XE -s 14 -h 206 -w 320 my130XEfile.cfg
```

Installation Etc.

Right now, it is assumed you know enough to install and run Python, install dependencies, and launch Python scripts/files from the command line.

If there is enough interest, I'll write up something for those not familiar with Python when I get a chance.

You'll require a current version of Python installed. I'm using version (3.12.2). I've not tested with any other version.

I've only run this on macOS.

It *should* work fine under Linux or Windows, but I've not tried it.

It is **STRONGLY** recommended that you create a Python virtual environment for this. One of the dependencies, "Pillow", will **not** work side-by-side with the standard Python image library (PIL) – so if you install "Pillow" in a shared/global environment you may break something that relies on PIL.

Dependencies are just Click.

You can install it with the command:

```
pip install -r requirements.txt
```