scriptabit

Release 2.1.2

Jun 07, 2018

Contents

Python scripting and scenarios for Habitica.

- Free software: Apache 2.0
- Homepage: https://github.com/DC23/scriptabit
- Documentation: https://scriptabit.readthedocs.org
- Version: 2.1.2

CHAPTER 1

Contents

1.1 Readme

Python scripting and scenarios for Habitica.

- Free software: Apache 2.0
- Homepage: https://github.com/DC23/scriptabit
- Documentation: https://scriptabit.readthedocs.org
- Version: 2.1.2

Note You can use the Github issues for bugs and feature requests, however most task and feature planning is carried out in a private Trello board. Access can be provided on request.

1.1.1 Installation

To install the latest release from PyPI:

```
pip install scriptabit
```

If you already have *scriptabit* installed, then upgrade with:

```
pip install --upgrade scriptabit
```

Habitica Credentials

You require an authentication credentials file in your home directory containing your Habitica API Key and User ID. The file should have a typical ini file structure, with the following section:

```
[habitica]
userid =
apikey =
```

Additional sections can be added, and the section name to use can be supplied as a command-line argument.

If you do not already have a *.auth.cfg* file, a default will be created when you first run scriptabit. You can then fill in your account values.

Once you have entered your Habitica credentials, test them with the *-sud* command (short for *–show-user-data*):

```
scriptabit -sud
```

If everything is set up correctly, you should see a summary of your character data printed to the console.

Note that your API key is effectively a password to your Habitica account. You should make sure the .auth.cfg file is protected, and never share the key with others. On Linux and related systems, you can set the permissions as follows:

```
chmod 600 .auth.cfg
```

Trello Credentials

If you wish to use the Trello plugin, you will need to add your Trello credentials to the .auth.cfg file as follows:

```
[trello]
apikey =
apisecret =
token =
tokensecret =
```

Your API key and API secret can be obtained here.

Your authorisation token and token secret will be obtained through an interactive process when you first run the trello plugin. You must first save your API key and API secret to the .auth.cfg file before you will be able to obtain the token and tokensecret.

1.1.2 Usage

scriptabit is a command-line application. Help on the available commands can be obtained by running:

```
$ scriptabit --help
```

Operations include:

- *-sud*: Show user data.
- *-hp n*: Set the user health to n
- *-mp n*: Set the user's mana points to n
- *-xp n*: Set experience points to n
- *-gp n*: Set gold to n
- *-ls*: List available plugins.

After running *scriptabit* at least once, configuration files will be created in *~/.config/scriptabit/*. These can be edited to change the default options. You can revert to the installation defaults by deleting the files (they will be recreated on the next run).

See the *Working with User Stats* section for detailed instructions on specific functionality.

Finally, most of the built-in plugins define a convenience command-line application name:

- *sb-banking* is a shortcut for *scriptabit –run banking*
- *sb-csv* is a shortcut for *scriptabit –run csv_tasks*
- *sb-health* is a shortcut for *scriptabit –run health_effects*
- *sb-pets* is a shortcut for *scriptabit –run pet_care*
- *sb-trello* is a shortcut for *scriptabit –run trello*
- *sb-tasks* is a shortcut for *scriptabit –run tasks*

When using the shortcuts, all other command-line arguments are the same as when running *scriptabit*.

Notification Panel

By default, most scriptabit operations update a scoreless habit in Habitica with some status information. This can be useful when you have some functions running in an update loop.

The use of this panel can be controlled with the `use-notification-panel` argument, either on the command line or by setting a value into the scriptabit.cfg file. Set to 0 or False to suppress the panel.

Habitica Tags

By default, scriptabit applies the *scriptabit* tag to all the tasks it creates in Habitica. This behaviour can be controlled with the `--tags` option. It accepts a comma-separated list of tags.

To disable the use of tags, set the option to an empty string: `--tags ""`

1.1.3 Writing Plugins

User plugins should be placed into the *scriptabit_plugins* directory. This will be created in your home directory the first time *scriptabit* runs. Due to an initialisation order issue, this directory location cannot be specified on the command line (the plugin directory needs to be located before processing command line arguments so that plugins get a chance to add additional arguments). If the *SCRIPTABIT_USER_PLUGIN_DIR* environment variable is defined, then this location will be used instead of the default location.

Note that plugin data files may also be written to the user plugin directory

All plugins should subclass the *IPlugin* class. Refer to the API documentation for details of the available methods.

Also refer to the API documentation (and the view source option) for the sample plugin which can be used as a template for new plugins.

1.2 Working with User Stats

Basic user stats can be displayed with the `--show-user-data` (short form `-sud`) command:

```
scriptabit -sud
```

Additionally, gold, mana, health, experience points, and character level can all be modified with `set`, `increment`, and `scale` commands. The `set` commands set the values directly, the `increment` commands add or subtract the specified amount from the current value, and the `scale` commands multiply the current value by the supplied scaling factor.

The `set` commands all have the form `--set-XX`, where XX is one of:

- gp: gold points
- hp: health points
- mp: mana points
- xp: experience points
- level: character level

The `increment` commands all have the form `--inc-XX`, using the same codes.

The `scale` commands all have the form `--scale-XX`, using the same codes.

1.2.1 Example 1: subtract 10 health points

Use an increment command for hp, and a value of -10 to reduce health by 10 points:

```
scriptabit --inc-hp -10
```

1.2.2 Example 2: Set gold to 1000

Use the set command for gp, and a value of 1000:

```
scriptabit --set-gp 1000
```

1.2.3 Example 3: Reduce experience points by half

Use a scale command on XP:

```
scriptabit --scale-xp 0.5
```

1.3 Using the Built-in Plugins

Some features of scriptabit are more complex than setting HP via the command line. These features are described in more detail here.

1.3.1 Banking

The banking plugin automates the Gold Bank custom reward, allowing you to check your balance, and make deposits or withdrawals.

The first time the banking plugin runs it will create a custom reward called *Gold Bank*. You can edit the name of the custom reward in Habitica once it has been created, but do not change the task alias.

Do not delete the custom reward without first withdrawing all the gold, or your gold will be lost.

Also, don't purchase the bank reward in Habitica. The current balance is shown as the reward value for convenience. Actually purchasing the reward in-game will cost gold equal to the bank balance, but will not update the actual balance (although you could do this manually if required - simply edit the extra notes field in the reward).

The primary advantage of using a bank is that the gold is not lost on death. However, to maintain some balance the scriptabit bank charges *Bank Fees* on each transaction.

Gold, Mana, and Health Banks

Since version 1.15.0, scriptabit banking supports mana and health banks in addition to the gold bank. These work identically to the gold bank except for the following:

- You must specify the bank type on the command line. `sb-banking --bank-type mana` or `sb-banking --bank-type health`.
- Each bank is stored in a separate custom reward.
- Fees are charged on a different scale. Health fees have a max of 5, and mana fees a max of 20. These are not adjustable through the program options.
- Your health cannot be set below 1 or above 50. In fact, due to the fees, you probably can't get health above 49.
- Mana withdrawals are capped so you cannot exceed your maximum mana (based on intelligence).
- The bank tax option is only available for the gold bank.

Checking your balance

The simplest way to check your balance is through one of the Habitica apps or the website. The current balance is stored in both the extra notes field, and in the reward value.

To check your balance with scriptabit:

```
sb-banking -b
```

There are no fees for balance checks.

Deposits

Deposits are made with the *–bank-deposit* argument:

```
scriptabit --run banking --bank-deposit 100
```

Or using the shortcut method (short-form arguments and banking command):

```
sb-banking -d 100
```

Deposits are capped by your available gold, so trying to deposit more gold than you have is a simple way to deposit all your gold.

Withdrawals

Withdrawals are made with the *–bank-withdraw* argument:

```
scriptabit --run banking --bank-withdraw 100
```

Or using the shortcut method (short-form arguments and banking command):

```
sb-banking -w 100
```

Withdrawals are capped by the bank balance, so trying to withdraw more gold than you have is a simple way to withdraw all gold from the bank.

Paying Taxes

Note that taxes only work for gold banks, not mana or health

The banking plugin allows you to pay taxes. The gold will be deducted first from your gold balance and then from the bank if required. This may be of use to players seeking an extra challenge:

```
scriptabit --run banking --bank-tax 100
```

Or using the shortcut method (short-form arguments and banking command):

```
sb-banking --bank-tax 100
```

Bank Fees

Extra realism is available through bank fees. Fees are charged for deposits and withdrawals. Small transactions are expensive, with larger transactions becoming better value for money. The fees level off significantly after 1000 gold. This means that the most cost effective way to use the bank is to save at least 1000 gold first, however this increases the risk of losing your gold due to death. It is up to you to balance the transaction cost with the risk of death.

The command line argument *bank-max-fee* sets the upper limit on fees. Values up to 600 will make transactions very expensive, while going beyond 600 will start to make small transactions cost more than the transaction amount.

Fees can be disabled by setting *bank-max-fee* to zero. This can be on the command line, or permanently by adding the following to the scriptabit.cfg file:

```
[banking]
bank-max-fee = 0
```

1.3.2 CSV Batch Upload

Example command line:

```
scriptabit --run csv_tasks --csv-file [my_csv_file]
```

Or using the csv-specific command:

```
sb-csv --csv-file [my_csv_file]
```

The *csv-tasks* plugin provides limited support for batch creation of Habitica tasks from a CSV file. The supported task features are:

- Create habits, dailies, todos, or rewards.
- task names
- task extra text
- difficulty level (trivial, easy, medium, or hard)

- character attribute (for attribute-based automatic levelling)
- tags
- for habits, the up and down scoring buttons can be specified.
- for rewards, the reward value can be specified.

Habitica task features that are **not currently supported** are:

- Due dates
- Checklists
- Repeat options (days of the week, every X days) for Dailies.

The CSV file must have a first row header, with the following columns:

- name (required): Values are used for the task name.
- type (required): Values must be either habit, daily, or todo.
- description (optional): Values used as the extra notes.
- difficulty (optional): The task difficulty. Values should be one of trivial, easy, medium, or hard.
- attribute (optional): Values should be one of strength, intelligence, constitution, or perception.
- tags (optional): Values should be a comma-separated list of tag names for each task. Leave the entry blank if you don't want tags applied to a task.
- up (optional): This value is only used for habits. Any text at all here means the habit will have an up button.
- down (optional): This value is only used for habits. Any text at all here means the habit will have an down button.
- value (optional): Only used for rewards. Must be an integer value that is greater than zero. Used to set the value of the custom reward.

1.3.3 Health Effects

The Health Effects plugin currently provides simple implementations of poisoning/health-draining and health regeneration. The longer-term plan for this plugin is to provide a more complex scenario of poisoning and health regeneration effects that are based on your performance on Dailies and Habits. Custom rewards will also be created to implement cures that can be purchased in game.

Examples

Drain 15 HP over 24 hours, updating every 30 minutes:

```
sb-health --update-frequency 30 --max-hp-change-per-day 15 --health-drain
```

Regenerate 10 HP over 24 hours, updating every 30 minutes (shortcut args):

```
sb-health -uf 30 -hp24 10 --health-regen
```

1.3.4 Pet Care

The Pet Care plugin streamlines a number of operations that are difficult to carry out in large numbers through the current Habitica applications. Core features are:

- *list-pets*: List current inventory of pet-related items. This is primarily for development support, as this task is easily achieved through the official applications.
- *feed-pets*: Batch pet feeding.
- *hatch-pets*: Batch pet hatching.

A number of flags exist to control the pets, food, and potions used for the batch operations. They are:

- *any-pet-food*: If supplied, then all food types will be offered to all pets. The default behaviour is to only use the preferred foods for a pet. The main use of this flag is for the case where you have just a few pets, but a lot of food. It will let you raise the pets to a mount more quickly without having to wait for the preferred foods.
- *no-base-pets*: If supplied, then the standard pets will be excluded from batch operations, allowing you to focus on quest and magic potion pets. The default is to only feed standard pets.
- *quest-pets*: If supplied, quest pets will be included in batch operations. The default is to exclude quest pets.
- *magic-pets*: If supplied, magic potion pets will be included in batch operations. The default is to exclude magic potion pets.
- *no-raise*: If supplied, pets will not be raised to mounts during feeding.

Note that apart from the flags described above, you cannot control the order in which pets are fed. Each pet will be repeatedly offered food until either the (preferred) food is all gone, or the pet becomes a mount. If more food or pets remain, then the process repeats. If you want to feed specific pets, this should be done through the official applications.

Examples

These example command lines all use the shortcut method. The long form would use *scriptabit –run pet_care* instead of *sb-pets*.

Feed preferred foods to standard pets only:

```
sb-pets --feed-pets
```

Feed preferred foods to standard pets but do not raise to mounts:

```
sb-pets --feed-pets --no-raise
```

Feed any food to quest pets only:

```
sb-pets --feed-pets --no-base-pets --quest-pets --any-pet-food
```

Hatch all available standard pets:

```
sb-pets --hatch-pets
```

Hatch all available standard and quest pets:

```
sb-pets --hatch-pets --quest-pets
```

1.3.5 Spell/Skill Casting Plugin

The `spellcast` plugin allows you to cast skills/spells. Some spells require a target (eg: the Rogue Backstab skill). For these skills, you need to supply the Habitica unique ID of the target (the UUID).

This plugin is primarily intended for use from scripts, as obtaining the required UUIDs and entering them on the command line is error prone and tedious.

Example command line:

```
sb-cast --cast-skill toolsOfTrade
```

You can obtain the UUID of a task by using the tasks plugin to list tasks with the `show-uuid` or `verbose` flags:

```
sb-task --list-tasks --show-uuid
```

If using the `verbose` flag, all information will be displayed. The `id` or `_id` field contains the required value.

For casting skills on other party members, you can obtain the player UUID from the Habitica web client.

Example showing backstab being cast three times on a task:

```
sb-cast --cast backStab --target 7b069a9e-80aa-47ab-99fd-15fae4dd8ba7 -n 3
```

Options are:

- `--preserve-user-hp`: For spells that modify health, this option will preserve the player's original health value. This only works with the Blessing skill (`healAll`), allowing the player to heal the party but not themselves.
- `-n N`: An optional count (defaults to 1). Skills will be cast N times, or until mana is exhausted.

1.3.6 Tasks Plugin

Example command line:

```
sb-tasks --list-tasks
```

The `tasks` plugin provides a collection of task manipulation functions. For batch upload of tasks from a CSV file, please see the `csv-tasks` plugin. The supported task features are:

- `--list-tasks` List the tasks. If the `--verbose` option is specified, then a full data dump of the tasks is made. Otherwise just the names are listed.
- `--delete-tasks` Deletes all tasks of the specified type.
- `--list-tags` List all tags.
- `--list-unused-tags` List all tags that are not assigned to any tasks.
- `--delete-unused-tags` Delete all tags that are not assigned to any tasks. Note that if the `dry-run` flag is also given, this will revert to simply listing the unused tags.

Options are:

- `--verbose`: Verbose output.
- `--show-uuid`: Show the task UUID when listing tasks.
- `--task-type`: Specify the type of task to operate on. Values are *habits*, *dailies*, *todos*, *rewards*, or *all*.
- `--dry-run`: List the actions that would be carried out, but don't change anything on the server.

1.3.7 Trello

The *trello* plugin provides synchronisation (currently only one-way) from Trello cards to Habitica todos. Key features are:

- Sync selected Trello boards and lists (selectable by name)
- Sync all cards on a board, or just those assigned to you (good for shared boards)
- Use Trello labels to control the Habitica todo difficulty and character attribute (useful when using automatic attribute point assignment). The required labels will be created if necessary by the trello plugin. The labels are created with no color in Trello so they won't interfere with any existing labels, but the color can be changed in Trello if desired.
- Use a 'no sync' label on cards to prevent synchronisation of that card even if it meets all the other criteria for synchronisation.
- The Python script can run in a loop, checking for changes every 30 minutes by default (but this is adjustable).
- The default task difficulty and character attribute can be set per board.
- Cards that have already been seen are remembered between runs, so after the first synchronisation the updates will be a lot faster.

The following features are synchronised on each card:

- Name.
- Description. Optional, off by default.
- Due date.
- Checklists.
- Difficulty (trivial, easy, medium, hard) via Trello labels.
- Character attribute.
- Completion status.

Configuration

Refer to the *Trello Credentials* section of the README for instructions on configuring authentication with your Trello account.

Although configuration options for boards and lists can be supplied on the command line, they are best set in the scriptabit configuration file. See the *Usage* section for general information on the configuration file, including its default location. Only Trello specific options are discussed here.

This is a typical configuration file section for the trello plugin:

```
[trello]
# Specify default settings for each board by specifying a composite string:
#    board_name|difficulty|attribute|user
# If you do not specify the difficulty then it defaults to easy.
# If you do not specify the attribute then it defaults to strength.
# If you don't specify anything for the user field, it defaults to using all
# cards on the board.
# E.G.:
trello-boards = [Work|easy|intelligence|user, scriptabit|easy|intelligence]
trello-lists = [Next Actions, Waiting For, Scheduled, Backlog, Doing]
trello-done-lists = [Done]
```

Boards

Boards are specified by name, with additional board options specified in the composite string. The format is *Board name|default difficulty|default attribute|user*. If the difficulty is not specified it defaults to easy, and if the attribute is not specified it defaults to strength. This matches the Habitica defaults.

Legal values for difficulty are:

- trivial
- easy
- medium
- hard

Legal values for attribute are:

- strength
- intelligence
- perception
- constitution

Invalid values will cause a warning, and the default values will be used instead.

Finally, note that the order of board options is fixed. So if you want to specify an attribute default, you must also specify the preceeding difficulty.

Lists

Lists are also specified by name. *trello-lists* defines the list names that will be searched for incomplete cards, while *trello-done-lists* defines the names of the lists that indicate completed cards. See *Done Lists versus Archiving: A Note on Task Completion in Trello* for details on the use of Done lists as opposed to archiving cards.

Done Lists versus Archiving: A Note on Task Completion in Trello

The concept of a card being completed doesn't really exist in Trello. Two possibilities of marking a Trello card as complete are archiving the card, or moving it to a Done list. For efficiency reasons, the scriptabit trello plugin requires that you use a Done list. Archived cards are never seen by scriptabit, so if you archive a card then the trello plugin will think the card has been deleted rather than completed. The key reason for this behavior is that Trello boards can easily have thousands of archived cards, and checking all of these on every synchronisation would be very slow. The Trello API does not supply a method to only retrieve cards that were archived within the last day or so. I have therefore chosen to treat a configurable list as indication that a card is "done". Once the cards in the Done list have been synchronised to Habitica they can then be archived in Trello without consequences.

Example Command Lines

Run the trello plugin with default options:

```
sb-trello
```

Run once rather than syncing every 30 minutes:

```
sb-trello --max-updates 1
```

1.4 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

1.4.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- The scriptabit package version.
- Detailed steps to reproduce the bug.

1.4.2 Documentation improvements

scriptabit could always use more documentation, whether as part of the official scriptabit docs, in docstrings, or even on the web in blog posts, articles, and such.

Note: This project uses Google-style docstrings. Contributed code should follow the same conventions. For examples, please see the Napoleon examples, or the Google Python Style Guide.

1.4.3 Feature requests and feedback

The best way to send feedback is to file an issue

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Or, implement the feature yourself and submit a pull request.

1.4.4 Development

Setting up your Git Repository

To set up *scriptabit* for local development:

1. Fork the *scriptabit* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/scriptabit.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

4. When you're done making changes, run all the tests, doc builder and pylint checks:

```
py.test
pylint ./src/scriptabit/
sphinx-build -b html docs build/docs
```

Or, using the project makefile:

```
make clean lint tests docs
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

The Development Environment

There is a makefile in the project root with targets for the most common development operations such as lint checks, running unit tests, building the documentation, and building installer packages. *tox* does not have a target, as *make tox* is more typing than *tox*.

Run make with no target to see the list of targets:

```
$ make
```

Bumpversion is used to manage the package version numbers. This ensures that the version number is correctly incremented in all required files. Please see the bumpversion documentation for usage instructions, and do not edit the version strings directly.

Version numbers follow the Semantic versioning guidelines.

I recommend using a Python virtual environment for development. Although not essential, it is helpful to isolate the project from other Python packages that may be installed. Once you create and activate the virtual environment, the command *make develop* will install all the development dependencies and *scriptabit* in develop mode. From then on, only changes to the setup files (such as adding new entry points) will require rerunning the *make develop* command. My method for setting up on Linux is basically:

```
$ git clone https://github.com/DC23/scriptabit.git
$ cd scriptabit
$ mkvirtualenv -a . -p /usr/bin/python3 scriptabit
$ workon scriptabit
$ make develop
$ make tests
$ tox
```

The last two lines run all the tests, both directly in the dev virtual environment and in tox against Python 2 and 3.

1.4.5 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `py.test`).
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

1.5 Authors

1.5.1 Development Lead

- DC23 <jugglindan@gmail.com>

1.5.2 Contributors

1.6 Changelog

1.6.1 0.1.0 (2016-07-29)

- First version
- Generated project boiler plate with cookiecutter and the cookiecutter-dcpypackage template.
- Adjusted the cookiecutter output.

1.6.2 0.2.0 (2016-08-03)

- First version that does anything useful :)
- Documentation cleaned up
- CI builds in Travis.
- Documentation on ReadTheDocs
- Authentication from .auth.cfg file
- Command-line interface established
- Unit tests with pytest and requests_mock
- Utility functions to set HP, MP, and XP.

1.6.3 0.2.2 (2016-08-07)

- Fixes issue #2 by adding a multipath search for configuration files.

1.6.4 0.2.3 (2016-08-07)

- Fixed issue #3 (broken Travis builds) by making setuptools bootstrap more robust.
- Cleaned up authentication file handling - generate default, more error checks.
- Moved utility function CLI arg definitions into the utility functions class.

1.6.5 0.3.0 (2016-08-10)

- Implemented plugin framework.
- Added test utility function. The functionality of this will change all the time, depending on my current test needs.
- Changed scenario text to reference plugins instead.
- Added max-updates command line argument.

1.6.6 0.4.0 (2016-08-11)

- Implemented banking plugin.
- Fixed issue where the app would sleep after updating a plugin even when no more updates were required.
- Added task upsert method to HabiticaService.

1.6.7 0.4.1 (2016-08-11)

- Updated documentation to include the built-in plugins.

1.6.8 0.4.2 (2016-08-11)

- Minor documentation fixes.

1.6.9 0.4.3 (2016-08-11)

- Restructured the documentation layout. I think this has better organisation of the indices and TOCs.

1.6.10 0.4.4 (2016-08-11)

- Updated status to alpha
- Fixed bug with bank task name being overwritten.
- Refactored HabiticaService upsert method
- Added create_task and get_task methods to HabiticaService

1.6.11 0.4.5 (2016-08-12)

- Added to PyPI
- Fixes issue #5 (missing files in PyPI package)

1.6.12 0.4.6 (2016-08-12)

- Fixes issue #6 (missing plugin metadata files in PyPI package)

1.6.13 0.4.7 (2016-08-12)

- Changed default logging options to be less intrusive.
- A lot of fixes to datetime handling, which should fix #7

1.6.14 0.5.0 (2016-08-16)

- Fixed load-order issue for configuration files. Who would have thought that configargparse started looking for configuration files from the end of the list rather than the start?
- Started adding Trello sync plugin. Doesn't really do much yet, but all the authentication song and dance code is in place, as well as configuration for defining which boards to sync, the lists on those boards, and the optional lists that will indicate task completion.
- Implemented core functionality for task synchronisation between generic task services.

1.6.15 1.0.0 (2016-08-17)

- Fixed task sync bug with missing destination tasks.
- HabiticaService now supports a task filter on get_tasks
- Implemented Habitica sync task
- Implemented HabiticaTaskService
- Updated documentation with sync information
- Implemented first version of Trello to Habitica sync. Tasks only, no difficulties, attributes, due-dates or check-lists.

1.6.16 1.1.0 (2016-08-17)

- Trello sync now uses optional Trello labels to indicate Habitica task difficulty and character attribute (strength, perception, intelligence, constitution). If they don't exist, the labels are created automatically.
- Habitica cards are now created with a "Trello" tag.
- Added due date synchronisation support.

1.6.17 1.2.0 (2016-08-18)

- Made sync data file name into a trello plugin argument
- Refactored task sync class into smaller functions that make the logic easier to read and modify.
- Added last_modified property to Task (and TrelloTask, HabiticaTask)
- Added modification time check for deciding whether to update tasks
- Added persistence for last sync time

- Improved sync stats reporting

1.6.18 1.3.0 (2016-08-19)

- Refined sync log message levels to reduce the spam at info levels
- Added notification ability by using a scoreless habit whose text can be updated by scriptabit functions.
- Added notifications to banking and trello plugins.
- Added global command-line argument for specifing the update interval of looping plugins.
- Implemented ability to set default difficulty and character attribute for cards on a Trello board.
- Implemented ability to sync all cards on a board, or just those assigned to the current user.
- Minor bug fixes, and lint warning cleanups.

1.6.19 1.4.0 (2016-08-22)

- Added checklist support to Trello sync
- Added CSV batch task creation plugin.
- Slightly improved error handling during task sync. Now an error in a task doesn't bring the whole sync down. Instead it logs the error and skips to the next task.
- Made sync of task description/extra text optional, with default to False.

1.6.20 1.5.0 (2016-08-24)

- Updated usage documentation for banking, and CSV upload.
- Added transaction fee option to banking functions.
- Added tax feature to banking plugin.
- Added direct gold amount setting to utility functions (-gp X)
- Changed bank fees to use a diminishing returns function, to reward the larger risk of saving longer to deposit larger amounts.

1.6.21 1.6.0 (2016-08-25)

- Added pet feeding function to pet care plugin.
- Trello cards that are both new and completed are now synchronised to Habitica if their last update time is more recent than the last synchronisation.
- Added support for a 'no sync' label on Trello cards. Cards with this label are ignored even if they meet all the other criteria for synchronisation.
- Added pet care usage documentation.

1.6.22 1.7.0 (2016-08-26)

- Added trello plugin usage documentation.
- Added pet hatching function to pet care plugin.
- Updated documentation.

1.6.23 1.7.1 (2016-08-29)

- Fixed #13: incorrect pet count when API error occurs.
- Made pet-care commands more logical (issue #16)
- Fixed issue #14: trying to feed pet when mount already exists.
- Fixed issue #15: errors if config and user plugin directories don't exist on first run (when depth is > 1)
- Added note to CSV plugin docs indicating that Daily repeat options are not supported.

1.6.24 1.8.0 (2016-08-29)

- Added messages and checks for dry run support in plugins.
- Implemented full dry run mode support in banking.
- Implemented full dry run mode support in CSV uploader.
- Implemented full dry run mode support in pet care plugin.
- Implemented full dry run mode support in trello sync plugin.
- Implemented full dry run mode support in the utility functions.
- Updated utility functions so they return the new value set into the character stats (gold, XP, HP, MP).

1.6.25 1.8.1 (2016-08-29)

- Added missing supports_dry_run method to pet care plugin, that was preventing dry runs.
- Fixed incorrect dryrun message in main loop.

1.6.26 1.9.0 (2016-08-30)

- Updated documentation.
- Added new entry points for built-in plugins:
  - *sb-banking* instead of *scriptabit –run banking*
  - *sb-trello* instead of *scriptabit –run trello*
  - *sb-pets* instead of *scriptabit –run pet_care*
  - *sb-csv* instead of *scriptabit –run csv_tasks*
  - *sb-health* instead of *scriptabit –run health_effects*

1.6.27 1.10.0 (2016-08-30)

- Made plugin update more robust. Exceptions are caught so that updates can continue rather than aborting the whole run.
- Implemented simple health drain and regeneration in health effects plugin.
- Refactored plugin notification methods.
- Changed pet care so it only sleeps during feeding if API calls were made during the last pet.

1.6.28 1.11.0 (2016-09-20)

- Minor logging changes: config file specifiable via environment variable (*SCRIPTABIT_LOGGING_CONFIG*). Log statement with the location of the user plugin directory.
- Added 10 second timeout to http requests.
- Added simple vampire mode - health drain during the day, slower regen at night.
- Added bank-balance option so that sb-banking with no args can display usage information.
- Changed main loop so that utility functions don't try to run at the same time as plugins.
- Removed upper MP limit check when setting mana.
- Added delete all todos utility function (can update to support other task types later, but todos is all I needed right now).
- Fixed bug where last Trello sync time was displayed in UTC rather than local time.

1.6.29 1.12.0 (2016-10-24)

- Updated pet care to handle the candy foods.
- Updated pet care to handle the spooky and ghost potions.
- Added purchase armoire item utility function, with optional repeat.

1.6.30 1.12.1 (2016-10-24)

- Fixed bug in *–any-pet-food* feeding option in pet-care plugin.

1.6.31 1.12.2 (2016-10-31)

- Added dry run support to armoire purchases
- Added 2 second delay between repeated armoire purchases.

1.6.32 1.12.3 (2016-11-25)

- Made timestamp optional for all notification messages.

1.6.33 1.13.0 (2016-12-02)

- Added spell/skill casting function.

1.6.34 1.13.1 (2016-12-02)

- Added list all tasks function

1.6.35 1.14.0 (2016-12-09)

- Added spellcast plugin

1.6.36 1.14.1 (2016-12-21)

- Added HP preservation option to spellcast plugin. This allows casting Blessing spells that don't heal the player.

1.6.37 1.15.0 (2016-12-22)

- Adding increment functions for XP, MP, HP. I already have direct set functions but the new ones are easier when you just want to increment the values up or down.
- Expanding the banking feature to allow mana and health banks as well as gold.
- Removed the bank name option, as it made supporting different bank types more complicated.

1.6.38 1.16.0 (sometime in January 2017)

- New tasks plugin. Run with `sb-tasks`.
- Added minimum requests version to setup.py.
- Added entry point for spellcasting plugin `sb-cast`.
- Cleaned up argument parsing, so running a plugin entry-point with the –help argument only shows help for that plugin.
- Running `scriptabit --help` now only shows help for the main application rather than all possible args from all plugins.
- Added documentation for spell casting.
- Added fallback help display so that plugins that fail to define a `print_help` function will use the default help display.

1.6.39 1.17.0 (Monday January 16, 2017)

- Changed pet_care logic so it detects quest and magic pets implicitly. This means I don't need to manually update the code for new quest or magic pets.
- Added tag management features to `sb-tasks`. You can now list tags, list unused tags, and delete unused tags. I didn't add a function to delete a tag by name as this is trivial to do in the web app.
- Fixed the issue in 1.16.0 where plugin names would log as “None”.

- Added sample task CSV file showing the required layout for `sb-csv`
- Updated documentation for pet care and tasks.

1.6.40 1.17.2 (Tuesday January 17, 2017)

- Very minor changes to PyPI metadata and README file.
- Made the update and creation of the notification panel optional. Use the *use-notification-panel* command line argument to control the behaviour.
- Made the task tags configurable. Use the `--tags` argument to pass a comma separated list of tags that will be used with any task created in Habitica. If you use `--tags ""` then no tags will be created or applied.

1.6.41 1.18.0 (Monday January 23, 2017)

- Implemented character level set and increment/decrement.

1.6.42 2.0.0 (Monday January 30, 2017)

- Bumped to major version 2 because of the breaking change in Trello sync.
- Fixed issues in trello sync. Unfortunately they require a complete reset of sync data. See the README.rst for details.
- CSV task uploader now recognizes "difficulty" as a column header, as well as "priority". Documentation changed to document "difficulty", as "priority" is deprecated and only preserved for backwards compatibility.
- Added scaling option for basic user stats.
- Added option to prevent pet raising to mounts during batch feeding.

1.6.43 2.0.1 (Wednesday February 01, 2017)

- Fixed the `no-raise` option for pet feeding.

1.6.44 2.0.2 (Thursday November 30, 2017)

- bidict compatibility fix from nightscape.

1.6.45 2.1.0 (Tuesday April 17, 2018)

- Fixed issue #44: no-raise option not working for pet feeding.
- Fixed issue #47: bad request caused by attempting to hatch a magic quest pet (this is an invalid combination).
- Fixed code for breaking API change in bidict (yes, another one).

1.6.46 2.1.1 (Tuesday April 17, 2018)

- Added code to prevent the attempt to delete tasks that belong to a challenge.

1.6.47 2.1.2 (Thursday June 07, 2018)

- Fixed issue #45: ethereal surge spell cast not working.

1.7 License

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

1.8 API Reference

1.8.1 Authentication

Authentication credential loading functions

`scriptabit.authentication.`**`load_habitica_authentication_credentials`**(*config_file_name=u'.auth.cfg'*, *section=u'habitica'*)

Loads authentication credentials from an ini-style configuration file.

Parameters

- **config_file_name** (*str*) – Basename of the configuration file.
- **section** (*str*) – Configuration file section name.

Returns the selected credentials

Return type dict

```
{
'x-api-user': 'the user name',
'x-api-key':  'the user API key',
}
```

Raises `ConfigError` – specified file section or options are missing

1.8.2 Configuration

Configuration and command-line arguments

`scriptabit.configuration.`**`__add_min_max_value`**(*parser*, *basename*, *default_min*, *default_max*, *initial*, *help_template*)

Generates parser entries for options with a min, max, and default value.

Parameters

- **parser** – the parser to use.
- **basename** – the base option name. Generated options will have flags –basename-min, –basename-max, and –basename.
- **default_min** – the default min value
- **default_max** – the default max value
- **initial** – the default initial value
- **help_template** – the help string template. $mmi will be replaced with min, max, or initial. $name will be replaced with basename.

`scriptabit.configuration.`**`copy_default_config_to_user_directory`**(*basename*, *clobber=False*, *dst_dir=u'~/.config/scriptabit'*)

Copies the default configuration file into the user config directory.

Parameters

- **basename** (*str*) – The base filename.
- **clobber** (*bool*) – If True, the default will be written even if a user config already exists.
- **dst_dir** (*str*) – The destination directory.

`scriptabit.configuration.`**`get_config_file`**(*basename*)

Looks for a configuration file in 3 locations:

- the current directory
- the user config directory (~/.config/scriptabit)
- the version installed with the package (using setuptools resource API)

Parameters **basename** (*str*) – The base filename.

Returns The full path to the configuration file.

Return type str

`scriptabit.configuration.`**`get_configuration`**(*basename=u'scriptabit.cfg'*, *parents=None*)

Parses and returns the program configuration options, taken from a combination of ini-style config file, and command line arguments.

Parameters

- **`basename`** (*`str`*) – The base filename.
- **`parents`** (*`list`*) – A list of ArgumentParser objects whose arguments should also be included in the configuration parsing. These ArgumentParser instances **must** be instantiated with the *add_help* argument set to *False*, otherwise the main ArgumentParser instance will raise an exception due to duplicate help arguments.

Returns The options object, and a function that can be called to print the help text.

1.8.3 Date Handling

Datetime functions

`scriptabit.dates.`**`parse_date_local`**(*date*, *milliseconds=True*)

Parses dates from ISO8601 or Epoch formats to a standard datetime object in the current local timezone.

Note that this function should not be used in time calculations. It is primarily intended for displaying dates and times to the user.

Parameters

- **`date`** (*`str`*) – A date string in either iso8601 or Epoch format.
- **`milliseconds`** (*`bool`*) – If True, then epoch times are treated as millisecond values, otherwise they are evaluated as seconds.

Returns The parsed date time in local time.

Return type datetime

`scriptabit.dates.`**`parse_date_utc`**(*date*, *milliseconds=True*)

Parses dates from ISO8601 or Epoch formats to a standard datetime object.

This is particularly useful since Habitica returns dates in two formats:

```
- iso8601 encoded strings
- Long integer Epoch times
```

Parameters

- **`date`** (*`str`*) – A date string in either iso8601 or Epoch format.
- **`milliseconds`** (*`bool`*) – If True, then epoch times are treated as millisecond values, otherwise they are evaluated as seconds.

Returns The parsed date time in UTC.

Return type datetime

1.8.4 Errors

Error classes

exception `scriptabit.errors.`**`ArgumentOutOfRangeError`**(*value*)
: A function argument is out of range

exception `scriptabit.errors.`**`ConfigError`**(*value*)
: Configuration file error

exception `scriptabit.errors.`**`InvalidHabiticaDataError`**(*value*)
: The specified Habitica data is invalid error

exception `scriptabit.errors.`**`NotFoundError`**(*value*)
: The specified Habitica item was not found error

exception `scriptabit.errors.`**`PluginError`**(*value*)
: Plugin error

exception `scriptabit.errors.`**`ServerUnreachableError`**(*value*)
: The Habitica server is unreachable

1.8.5 Habitica Service

class `scriptabit.`**`HabiticaService`**(*headers*, *base_url*)
: Habitica API service interface.

`buy_armoire`()
: Buy an armoire item.

Returns The Habitica response data.

Return type dict

`cast_skill`(*spellId*, *targetId=None*)
: Cast a skill.

Parameters

- **`spellId`** (*`SpellIDs`*) – The spell ID
- **`targetId`** (*`UUID`*) – Optional UUID of the spell target. Required for targetted spells.

Returns The Habitica response data.

Return type dict

`cast_skill_by_raw_spell_id`(*spellId*, *targetId=None*)
: Cast a skill using the raw Habitica API spell ID rather than the enum.

Parameters

- **`spellId`** (*`str`*) – The spell ID
- **`targetId`** (*`UUID`*) – Optional UUID of the spell target. Required for targetted spells.

Returns The Habitica response data.

Return type dict

`create_checklist_item`(*task_id*, *item*)
: Add a checklist item to the task.

Parameters

- **`task_id`** (*`str`*) – The task ID.
- **`item`** (*`dict`*) – The new checklist item.

`create_tag`(*name*)
Create a tag.

Parameters **`name`** (*`str`*) – the tag name.

Returns The new tag.

Return type dict

`create_tags`(*tags*)
Create the tags. Existing tags are ignored.

Parameters **`tags`** (*`list`*) – The list of tag names.

Returns The list of Habitica Tag objects corresponding to the tags argument.

Return type list

`create_task`(*task*, *task_type=<HabiticaTaskTypes.todos: u'todos'>*)
Creates a task.

Parameters

- **`task`** (*`dict`*) – The task.
- **`task_type`** (*`HabiticaTaskTypes`*) – The type of task to create. Default is to create a new todo. Only used if the task['type'] is empty or not present.

Returns The new task as returned from the server.

Return type dict

`create_tasks`(*tasks*)
Creates multiple tasks.

Note that unlike HabiticaService.create_task, this method **does not** check that the task type is valid.

Parameters **`task`** (*`list`*) – The list of tasks.

Returns The new tasks as returned from the server.

Return type list

`delete_checklist_item`(*task_id*, *item_id*)
Delete a checklist item.

Parameters

- **`task_id`** (*`str`*) – The task ID.
- **`item_id`** (*`str`*) – The checklist item ID.

`delete_tags`(*tags*)
Delete a list of tag objects.

Parameters **`tags`** (*`list`*) – The list of tag objects.

`delete_task`(*task*)
Delete a task.

Parameters **`task`** (*`dict`*) – The task.

`feed_pet`(*pet*, *food*)
Feed a pet.

Parameters

- **pet** (*str*) – The pet name.
- **food** (*str*) – The food.

Returns The Habitica response data.

Return type dict

get_stats()

Gets the authenticated user stats.

Returns The stats.

Return type dict

get_tags()

Get the current user's tags.

Returns The tags.

Return type list

get_task(*_id=u"*, *alias=u"*)

Gets a task.

If both task ID and alias are specified, then the ID is used.

Parameters

- **_id** (*str*) – The task ID.
- **alias** (*str*) – The task alias.

Returns The task, or None if the task is not found.

Return type dict

Raises `ValueError`

get_tasks(*task_type=None*)

Gets all tasks for the current user.

Parameters **task_type** (*HabiticaTaskTypes*) – The type of task to get. Default is all tasks apart from completed todos.

Returns The tasks.

Return type dict

get_user()

Gets the authenticated user data.

Returns The user data.

Return type dict

hatch_pet(*egg*, *potion*)

Hatch a pet.

Parameters

- **egg** (*str*) – The egg name.
- **potion** (*str*) – The potion name.

Returns The Habitica response data.

Return type dict

is_server_up()

Check that the Habitica API is reachable and up

Returns *True* if the server is reachable, otherwise *False*.

Return type bool

score_task(*task*, *direction=u'up'*)

Score a task.

Parameters

- **task** (*dict*) – the task to score.
- **direction** (*str*) – 'up' or 'down'

Returns Habitica API response data.

Return type dict

Raises

- `ValueError` – invalid direction.
- `ValueError` – missing ID or alias.

set_exp(*exp*)

Sets the user's XP (experience points).

Parameters **exp** (*float*) – The new XP value.

Returns The new XP value, extracted from the JSON response data.

Return type float

set_gp(*gp*)

Sets the user's gold (gp).

Parameters **gp** (*float*) – The new gold value.

Returns The new gold value, extracted from the response data.

Return type float

set_hp(*hp*)

Sets the user's HP.

Parameters **hp** (*float*) – The new HP value.

Returns The new HP value, extracted from the JSON response data.

Return type float

set_lvl(*lvl*)

Sets the user's character level. Note that XP will be reset to 0.

Parameters **lvl** (*int*) – The new level.

Returns The new character level, extracted from the JSON response data.

Return type lvl

set_mp(*mp*)

Sets the user's MP (mana points).

Parameters **mp** (*float*) – The new MP value.

Returns The new MP value, extracted from the JSON response data.

Return type float

update_task(*task*)

Updates an existing task.

Parameters **task** (*dict*) – The task.

Returns The new task as returned from the server.

Return type dict

Raises `ValueError` – if neither an ID or alias are present in task.

upsert_task(*task*, *task_type=<HabiticaTaskTypes.todos: u'todos'>*)

Upserts a task.

Existing tasks will be updated, otherwise a new task will be created.

Parameters

- **task** (*dict*) – The task.
- **task_type** (*HabiticaTaskTypes*) – The type of task to create if a new task is required. Can be overriden by an existing task['type'] value.

Returns The new task as returned from the server.

Return type dict

Raises `ValueError`

1.8.6 Plugin Baseclass

class scriptabit.**IPlugin**

Scriptabit plugin base class.

_config

lookupdict – Configuration object returned from argparse.

_update_count

int – Number of updates (zero-based).

_hs

scriptabit.HabiticaService – The HabiticaService instance.

activate()

Called by the plugin framework when a plugin is activated.

deactivate()

Called by the plugin framework when a plugin is deactivated.

dry_run

Indicates whether this is a dry run or not.

Returns True if this is a dry run, otherwise False.

Return type bool

get_arg_parser()

Gets the argument parser containing any CLI arguments for the plugin.

Note that to avoid argument name conflicts, only long argument names should be used, and they should be prefixed with the plugin-name or unique abbreviation.

To get their *ArgParser*, subclasses should call this method via super and capture the returned *ArgParser* instance.

Returns: argparse.ArgParser: The *ArgParser* containing the argument definitions.

`initialise`(*configuration*, *habitica_service*, *data_dir*)
Initialises the plugin.

Generally, any initialisation should be done here rather than in activate or __init__.

Parameters

- **`configuration`** (`ArgParse.Namespace`) – The application configuration.
- **`habitica_service`** – the Habitica Service instance.
- **`data_dir`** (`str`) – A writeable directory that the plugin can use for persistent data.

`notify`(*message*, ***kwargs*)
Notify the Habitica user.

If this is a dry run, then the message is logged. Otherwise the message is logged and posted to the Habitica notification panel.

Parameters

- **`message`** (`str`) – The message.
- **`panel`** (`bool`) – If True, the Habitica panel is updated.
- **`notes`** (`str`) – the extra text/notes.
- **`heading_level`** (`int`) – If > 0, Markdown heading syntax is prepended to the message text.
- **`alias`** (`str`) – the notification alias.

`static supports_dry_runs`()
Indicates whether the plugin correctly supports the *dry-run* command-line flag.

To support dry runs, the plugin must not modify any persistent data, either on Habitica or elsewhere if the *dry-run* flag is True.

Returns True if dry runs are supported, otherwise False.

Return type bool

`update`()
This update method will be called once on every update cycle, with the frequency determined by the value returned from *update_interval_minutes()*.

If a plugin implements a single-shot function, then update should return *False*.

Returns True if further updates are required; False if the plugin is finished and the application should shut down.

Return type bool

`update_interval_minutes`()
Indicates the required update interval in minutes.

Returns The required update interval in minutes.

Return type float

`update_interval_seconds`()
Indicates the required update interval in integer seconds.

Returns update interval in whole seconds

Return type int

1.8.7 Scriptabit

Scriptabit: Python scripting for Habitica.

`scriptabit.scriptabit.`**`start_scriptabit`**`()`
Command-line entry point for scriptabit

`scriptabit.scriptabit.`**`__init_logging`**`(`*logging_config_file*`)`
Initialises logging.

Parameters **`logging_config_file`** (*`str`*) – The logging configuration file.

`scriptabit.scriptabit.`**`__get_configuration`**`(`*plugin=None*`)`
Builds and parses the hierarchical configuration from environment variables, configuration files, command-line arguments, and argument defaults.

Parameters **`plugin`** – The optional plugin. If supplied, then the plugin arguments will be added to the parsed arguments.

Returns The argparse compatible configuration object and the help function

`scriptabit.scriptabit.`**`__get_plugin_manager`**`()`
Discovers and instantiates all plugins, returning a management object.

Returns The plugin manager with the loaded plugins.

Return type yapsy.PluginManager

`scriptabit.scriptabit.`**`__list_plugins`**`(`*plugin_manager*`)`
Lists the available plugins.

Parameters

- **`plugin_manager`** (*`yapsy.PluginManager`*) – the plugin manager containing
- **`plugins.`** (*`the`*) –

1.8.8 Synchronisation

Synchronisation services, classes, and interfaces, including Habitica classes. The remaining classes for synchronisation with specific services must be provided through plugins.

Task

Defines an abstract task.

`class` `scriptabit.task.`**`CharacterAttribute`**
Implements Task character attributes

`class` `scriptabit.task.`**`ChecklistItem`**`(`*name*, *checked=False*`)`
Simple implementation of a checklist item.

`__init__``(`*name*, *checked=False*`)`
Initialise the checklist item.

Parameters

- **name** (*str*) – The item name.
- **checked** (*bool*) – The item check-status.

class scriptabit.task.**Difficulty**
Implements Task difficulty levels.

class scriptabit.task.**SyncStatus**
Indicates the synchronisation status of the task.

class scriptabit.task.**Task**
Defines a Habitica task data transfer object.

Essentially this is the common features of a task as found in many task management applications. If a task from a particular service can be mapped to this class, then moving tasks between services becomes easier.

id
str – The task ID.

name
str – The task name.

due_date
datetime – The due date in UTC

description
str – The description.

completed
bool – Completion/checked status

difficulty
Difficulty – task difficulty.

attribute
CharacterAttribute – habitica character attribute of the task

status
SyncStatus – Synchronisation status flag.

checklist
list – The task checklist, or None if the task does not have a checklist.

__init__()
Initialise the task.

attribute
Task character attribute

checklist
The checklist.

Returns

The checklist, or an empty list if there are no checklist items.

Return type list

completed
Task completed

copy_fields(*src*, *status=<SyncStatus.updated: 2>*)
Copies fields from src.

Parameters

- **src** (Task) – the source task
- **status** (SyncStatus) – the status to set

Returns self

Return type *Task*

description
Task description

difficulty
Task difficulty

due_date
The due date in UTC if there is one, or None.

id
Task id

last_modified
The last modified timestamp in UTC.

name
Task name

status
Task status

TaskMap

Defines persistent 1-1 task mappings.

class scriptabit.task_map.**TaskMap**(*filename=None*)
Persistent 1-1 task mapping.

_TaskMap__map(*a*, *b*)
Associate two ID strings

__init__(*filename=None*)
Initialise the TaskMap instance.

Parameters **filename** (*str*) – The optional filename to load from.

get_all_dst_keys()
Gets a list of all destination keys.

Returns all destination keys.

Return type list

get_all_src_keys()
Gets a list of all source keys.

Returns all source keys.

Return type list

get_dst_id(*_id*)
Get the mapped destination task ID for a source task.

Parameters **_id** – The source task ID.

Returns If a mapping exists, the destination task ID.

Raises `KeyError` – if the input ID has no mapping.

`get_src_id`(*_id*)
Get the mapped source task ID for a destination task.

Parameters **`_id`** – The destination task.

Returns If a mapping exists, the source task ID.

Raises `KeyError` – if the input ID has no mapping.

`map`(*src*, *dst*)
Create a mapping between a source and destination task.

Parameters

- **`src`** (`Task`) – The source task.
- **`dst`** (`Task`) – The destination task.

`persist`(*filename*)
Persist the TaskMap instance to a file.

Parameters **`filename`** (*`str`*) – The destination file name.

`try_get_dst_id`(*_id*)
Get the mapped destination task ID for a source task.

Parameters **`_id`** – The source task ID

Returns If a mapping exists, the destination task ID, otherwise False.

Return type str

`try_get_src_id`(*_id*)
Get the mapped source task ID for a destination task.

Parameters **`_id`** – The destination task ID

Returns If a mapping exists, the source task ID, otherwise False.

Return type str

`unmap`(*src_id*)
Delete a mapping.

Parameters **`src_id`** – The source id to unmap.

TaskService

Defines an abstract task service.

A task service provides the following features:

- Query for tasks (including all tasks)
- Create a new task
- Persist a list of tasks

class `scriptabit.task_service.`**`TaskService`**
Defines an abstract Task Service.

`_create_task`(*src=None*)
Task factory method.

Allows subclasses to create the appropriate Task type.

Parameters **src** (*Task*) – The optional data source.

Returns The new task

Return type *Task*

create(*src=None*)
Creates a new task.

Parameters **src** (*Task*) – The optional data source.

Returns The new task.

Return type *Task*

get_all_tasks()
Get all tasks.

Returns The list of tasks

Return type list

persist_tasks(*tasks*)
Persists the tasks.

Parameters **tasks** (*list*) – The collection of tasks to persist.

TaskSync

Provides synchronisation between two task services.

Basic algorithm

- Build list of candidate tasks from source and destination services
- Index the candidate tasks for lookup by ID
- Get the existing list of source to destination task mappings
- Check all source tasks
  - Mapping exists, destination task found:
    - update destination
  - Mapping exists, destination task not found:
    - recreate destination
    - alternatively, could delete source task
  - No mapping found:
    - new task
    - if task completed, check if last modified date is newer than last sync date
- Check all destination tasks for which mapped source tasks can't be found:
  - assume deleted and flag destination as 'deleted'
- Check for orphan mappings: both source and destination not found
  - delete mapping
- **Not implemented**: persist source tasks

- Persist destination tasks

class scriptabit.task_sync.**TaskSync**(*src_service*, *dst_service*, *task_map*, *last_sync=None*, *sync_description=True*)

Provides synchronisation between two task services.

class Stats

Simple sync stats

__init__()

Initialise the stats

total_changed

Get the total number of changed tasks.

_TaskSync__clean_orphan_task_mappings()

Removes task mappings where neither the source or destination tasks exist.

_TaskSync__create_new_dst(*src*)

Creates and maps a new destination task.

Parameters **src** (Task) – source task

Returns The new destination task

Return type *Task*

_TaskSync__get_dst_by_id(*_id*)

Looks up a cached destination task by ID

_TaskSync__get_src_by_id(*_id*)

Looks up a cached source task by ID

_TaskSync__get_task_data()

Gets, caches, and indexes task data from the source and destination services.

_TaskSync__handle_deleted_source_task(*src_id*, *dst*)

Handle the case where a mapped destination task exists but the source task cannot be located.

Parameters

- **src_id** (*str*) – the source task ID
- **dst** (Task) – the destination task

_TaskSync__handle_destination_found(*src*, *dst*)

Handle the case where a pair of mapped tasks exist.

Parameters

- **src** (Task) – the source task
- **dst** (Task) – the destination task

_TaskSync__handle_destination_missing(*src*)

Handle the case where a mapped destination task cannot be found.

Parameters **src** (Task) – the source task

_TaskSync__handle_new_task(*src*)

Handle a new source task.

Parameters **src** (Task) – the source task

__init__(*src_service*, *dst_service*, *task_map*, *last_sync=None*, *sync_description=True*)

Initialise the TaskSync instance.

Parameters

- **`src_service`** (`TaskService`) – The TaskService for source tasks.
- **`dst_service`** (`TaskService`) – The TaskService for destination tasks.
- **`task_map`** (`TaskMap`) – The TaskMap.
- **`last_sync`** (*`datetime`*) – The last known synchronisation datetime (UTC).
- **`sync_description`** (*`bool`*) – Controls whether the task description will be synchronised.

`last_sync`

Gets the last synchronisation datestamp.

Returns The last synchronisation time.

Return type datetime

`synchronise`(*clean_orphans=False*)

Synchronise the source service with the destination. The task_map will be updated.

Parameters **`clean_orphans`** (*`bool`*) – If True, mappings for tasks that exist in neither the source or destination are deleted.

Returns Summary statistics of the sync.

Return type *TaskSync.Stats*

HabiticaTask

Implements a Habitica synchronisation task.

class `scriptabit.habitica_task.`**`HabiticaTask`**(*task_dict=None*)

Defines a Habitica synchronisation task.

`__init__`(*task_dict=None*)

Initialise the task.

Parameters **`task_dict`** (*`dict`*) – The Habitica task dictionary, as returned by HabiticaService.

`attribute`

Task character attribute

`checklist`

The checklist.

Returns

The checklist, or an empty list if there are no checklist items.

Return type list

`completed`

Task completed

`description`

Task description

`difficulty`

Task difficulty

due_date
The due date if there is one, or None.

id
Task id

is_challenge
Returns True is this is a challenge task.

last_modified
The last modified timestamp in UTC.

name
Task name

task_dict
Gets the internal task dictionary.

HabiticaTaskService

Implements the Habitica synchronisation task service.

class scriptabit.habitica_task_service.**HabiticaTaskService**(*hs*, *dry_run=False*, *tags=None*)
Implements the Habitica synchronisation task service.

_HabiticaTaskService__update_task(*task*)
Updates a task. This is required as checklists require tedious handling.

Parameters **task** (*scriptabit.HabiticaTask*) – The task to update.

__init__(*hs*, *dry_run=False*, *tags=None*)
Initialises the Habitica synchronisation task service.

Parameters

- **hs** (HabiticaService) – The Habitica Service.
- **dry_run** (*bool*) – Indicates a dry run.
- **tags** (*list*) – The list of tags to be applied to synchronised tasks.

_create_task(*src=None*)
Task factory method.

Parameters **src** (Task) – The optional data source.

Returns A new HabiticaTask instance.

Return type *HabiticaTask*

dry_run
Returns the dry run status.

get_all_tasks()
Get all tasks.

Returns The list of tasks

Return type list

persist_tasks(*tasks*)
Task factory method.

Allows subclasses to create the appropriate Task type.

Returns The new task

Return type *Task*

1.8.9 Utility Functions

class scriptabit.**UtilityFunctions**(*config*, *habitica_service*)

scriptabit utility functions. These are a collection of higher-level functions, implemented over the *HabiticaService* class.

__config

lookupdict – Configuration object returned from argparse.

__hs

scriptabit.HabiticaService – The HabiticaService instance.

buy_armoire()

Purchase an item from the Enchanted Armoire

dry_run

Indicates whether this is a dry run or not.

Returns True if this is a dry run, otherwise False.

Return type bool

get_arg_parser()

Gets the argument parser containing Utility function CLI arguments.

Returns The ArgParser containing the argument definitions.

Return type argparse.ArgParser

run()

Runs the user-selected scriptabit utility functions

set_gold(*gp*, *increment=False*, *scale=False*)

Sets the user gold to the specified value

Returns The new gold points.

Return type float

set_health(*hp*, *increment=False*, *scale=False*)

Sets the user health to the specified value

Returns The new health points.

Return type float

set_level(*level*, *increment=False*, *scale=False*)

Sets the user level to the specified value

Returns The new gold points.

Return type float

set_mana(*mp*, *increment=False*, *scale=False*)

Sets the user mana to the specified value

Returns The new mana points.

Return type float

set_xp(*xp*, *increment=False*, *scale=False*)
Sets the user experience points to the specified value.

Returns The new experience points.

Return type int

show_user_data()
Shows the user data

static upsert_notification(*habitica_service*, *text*, *notes=u''*, *heading_level=0*, *append_time=True*, *tags=None*, *alias=u'scriptabit_notification_panel'*)
Creates or updates a notification (currently implemented as a scoreless habit).

Parameters

- **habitica_service** (`HabiticaService`) – The habitica service to use.
- **text** (*str*) – the new text.
- **notes** (*str*) – the extra text/notes.
- **heading_level** (*int*) – If > 0, Markdown heading syntax is prepended to the message text.
- **append_time** (*bool*) – If True, a time stamp is appended to text.
- **tags** (*list*) – Optional list of tags to be applied to the notification.
- **alias** (*str*) – the notification alias.

Returns The notification object returned by the Habitica API

Return type dict

1.8.10 Bundled Plugins

Banking

Scriptabit plugin that implements a banking feature. Allows deposits and withdrawals from a custom bank.

If neither a deposit or withdrawal is specified, then the balance is reported but not changed.

Deposits and withdrawals are capped to the amount available, so a simple way to deposit or withdraw all the gold is to specify an amount larger than the balance.

class scriptabit.plugins.banking.**Banking**
Implements the banking plugin.

__init__()
Initialises the Banking instance.

calculate_fee(*amount*)
Calculates the fee for a given transaction amount.

Parameters **amount** (*float*) – The transaction amount.

Returns the transaction fee.

Return type float

deposit()
Deposit money to the bank.

get_arg_parser()
Gets the argument parser containing any CLI arguments for the plugin.

static **get_balance_from_string**(*s*)
Gets the bank balance from the formatted string

static **get_balance_string**(*amount*)
Gets the formatted bank balance string for a given amount

initialise(*configuration*, *habitica_service*, *data_dir*)
Initialises the banking plugin.

Parameters

- **configuration** (`ArgParse.Namespace`) – The application configuration.
- **habitica_service** – the Habitica Service instance.
- **data_dir** (`str`) – A writeable directory that the plugin can use for persistent data.

pay_tax()
Pays taxes, trying first from the main balance, and then from the bank.

static **supports_dry_runs**()
The Banking plugin supports dry runs.

Returns True

Return type bool

update()
Update the banking plugin.

Returns: bool: False

update_bank_balance(*new_balance*)
Updates the bank balance.

Parameters **new_balance** (`float`) – The new balance.

withdraw()
Withdraw money from the bank.

CSV Bulk Importer

Bulk creation of Habitica tasks from CSV files.

class scriptabit.plugins.csv_tasks.**CsvTasks**
Scriptabit batch CSV task importer for Habitica

_CsvTasks__fill_tag_placeholders()
Replace tag name placeholders with tag IDs

static **_CsvTasks__parse_bool**(*csv_value*)
parse a bool from a string value

static **_CsvTasks__parse_enum**(*enum*, *name*)
Parse an enum, trying both lookup by name and value. Returns the default if neither lookup succeeds.

__init__()
Initialises the plugin. Generally nothing to do here other than initialise any class attributes.

activate()
Called by the plugin framework when a plugin is activated.

deactivate()
Called by the plugin framework when a plugin is deactivated.

get_arg_parser()
Gets the argument parser containing any CLI arguments for the plugin.

Note that to avoid argument name conflicts, only long argument names should be used, and they should be prefixed with the plugin-name or unique abbreviation.

Returns: argparse.ArgParser: The *ArgParser* containing the argument definitions.

initialise(*configuration*, *habitica_service*, *data_dir*)
Initialises the plugin.

Generally, any initialisation should be done here rather than in activate or __init__.

Parameters

- **configuration** (*ArgParse.Namespace*) – The application configuration.
- **habitica_service** – the Habitica Service instance.
- **data_dir** (*str*) – A writeable directory that the plugin can use for persistent data.

static supports_dry_runs()
The CSV plugin supports dry runs.

Returns True

Return type bool

update()
This update method will be called once on every update cycle, with the frequency determined by the value returned from *update_interval_minutes()*.

If a plugin implements a single-shot function, then update should return *False*.

Returns: bool: True if further updates are required; False if the plugin is finished and the application should shut down.

Health Effects

Scriptabit plugin that implements various health-modification effects.

class scriptabit.plugins.health_effects.**HealthEffects**
Implements the health effects plugin.

__init__()
Initialises the plugin.

apply_health_delta(*hp24=None*, *up=True*)
Applies the health delta.

Parameters

- **hp24** (*float*) – The health change per 24 hours.
- **up** (*bool*) – If True, then health is increased, otherwise health is decreased.

Returns the signed health delta that was applied.

Return type float

get_arg_parser()
Gets the argument parser containing any CLI arguments for the plugin.

get_health_delta(*hp24=None*)
Gets the health delta for the current update

initialise(*configuration*, *habitica_service*, *data_dir*)
Initialises the plugin.

Parameters

- **configuration** (*ArgParse.Namespace*) – The application configuration.
- **habitica_service** – the Habitica Service instance.
- **data_dir** (*str*) – A writeable directory that the plugin can use for persistent data.

logistic_growth(*x*, *a=50*, *b=0.5*, *k_x_positive=0.2*, *k_x_negative=4.8*)
Returns the logistic growth function value for a given input x.

y = a / (1 + b * e^(kx))

For k > 0, larger values give a greater rate of change while smaller values lead to a slower approach to the function asymptotes.

For positive k, the return values is bounded by a to the left (large negative x) and by 0 to the right (large positive x). For negative k this is reversed.

Additionally, this method allows different k terms for positive and negative x, which allows finer-grained tuning of the output.

The y-intercept is given by a / (1 + b).

Parameters

- **x** (*float*) – The input value
- **a** (*float*) – The upper limit on the function value.
- **b** (*float*) – Influences the steepness of the function curve, and also contributes to the y-intercept. High values give a slower change and a lower y-intercept.
- **k_x_positive** (*float*) – The k term used when x >= 0
- **k_x_negative** (*float*) – The k term used when x < 0. Passing None will cause k_x_positive to be used for all x.

Returns The delta.

Return type float

poisoned()
Simple health drain/poisoning

regenerating()
Simple health regeneration

summarise_task_performance(*tasks*, *window_hours=24*)
Summarises overall task performance within a time window back from the current time.

Parameters

- **tasks** (*list*) – The list of Habitica tasks to summarise.
- **window_hours** (*float*) – Size of the time window in hours

Returns:

summarise_task_score(*task*, *now*, *window*)
Summarises the task score changes within a time window.

Parameters

- **`task`** (*`dict`*) – The task.
- **`now`** (*`datetime`*) – The most recent time to consider.
- **`window`** (*`timedelta`*) – The time window to consider prior to now.

Returns Total of the score changes within the time window. int: Number of times the score went up. int: Number of times the score went down.

Return type float

`static supports_dry_runs``()`
The HealthEffects plugin supports dry runs.

Returns True

Return type bool

`update``()`
Update the health effects plugin.

Returns: bool: True if further updates are required; False if the plugin is finished and the application should shut down.

`vampire``()`
Vampire mode.

Lose health during daylight hours. Gain small amounts of health at night, and large amounts by feeding.

Pet Care

Habitica pet care.

Options for batch hatching and feeding pets.

`class` `scriptabit.plugins.pet_care.`**`PetCare`**
Habitica pet care

`__init__``()`
Initialises the plugin.

`consume_food`(*food*)
Consumes a food, updating the cached quantity.

Parameters **`food`** (*`str`*) – The food.

`feed_pets``()`
Feeds all current pets.

`get_arg_parser``()`
Gets the argument parser containing any CLI arguments for the plugin.

Returns: argparse.ArgParser: The *ArgParser* containing the argument definitions.

`get_eggs`(*base=True*, *quest=False*)
Gets the filtered dictionary of available eggs. Values indicate current quantity.

Parameters

- **`base`** (*`bool`*) – Includes or excludes standard eggs.
- **`eggs`** (*`bool`*) – Includes or excludes quest eggs.

Returns The dictionary of eggs and quantities.

Return type dict

`get_food_for_pet`(*pet*)
Gets a food item for a pet

Parameters **`pet`** (*`str`*) – The composite pet name (animal-potion)

Returns

The name of a suitable food if that food is in stock. If no suitable food is available, then None is returned.

Return type str

`get_hatching_potions`(*base=True*, *magic=False*)
Gets the filtered dictionary of available hatching potions. Values indicate current quantity.

Parameters

- **`base`** (*`bool`*) – Includes or excludes standard potions.
- **`magic`** (*`bool`*) – Includes or excludes magic potions.

Returns The dictionary of potions and quantities.

Return type dict

`get_pets`(*base=True*, *magic=False*, *quest=False*, *rare=False*, *feedable_only=False*)
Gets a filtered list of current user pets.

Parameters

- **`base`** (*`bool`*) – Includes or excludes base pets.
- **`magic`** (*`bool`*) – Includes or excludes magic pets.
- **`quest`** (*`bool`*) – Includes or excludes quest pets.
- **`rare`** (*`bool`*) – Includes or excludes rare pets.
- **`feedable_only`** (*`bool`*) – If true, only feedable pets are included. Pets where a matching mount exists are not feedable.

Returns the filtered pet list.

Return type list

`has_any_food`()
Checks whether any food is left.

Returns True if some food remains, otherwise False.

Return type bool

`has_food`(*food*)
Returns True if the food is in stock, otherwise False.

Parameters **`food`** (*`str`*) – The food to check

Returns True if the food is in stock, otherwise False.

Return type bool

`hatch_pets`()
Hatch all available pets.

initialise(*configuration*, *habitica_service*, *data_dir*)
Initialises the plugin.

Generally, any initialisation should be done here rather than in activate or __init__.

Parameters

- **configuration** (*ArgParse.Namespace*) – The application configuration.
- **habitica_service** – the Habitica Service instance.
- **data_dir** (*str*) – A writeable directory that the plugin can use for persistent data.

is_base_pet(*pet*)
Is this a base pet?

Parameters **pet** (*str*) – The full pet name.

is_magic_pet(*pet*)
Is this a magic pet?

Parameters **pet** (*str*) – The full pet name.

is_magic_potion(*potion*)
Is this a magic potion?

In the current API, any potion not in the core list of standard potions is a magic potion.

Parameters **potion** (*str*) – The potion name.

is_quest_egg(*egg*)
Is this a quest egg?

In the current API, magic and base pets share the same core set of eggs, so anything not in that list is assumed to be a quest egg.

Parameters **egg** (*str*) – The egg name.

is_quest_pet(*pet*)
Is this a quest pet?

Parameters **pet** (*str*) – The full pet name.

is_rare_pet(*pet*)
Is this a rare pet?

Parameters **pet** (*str*) – The full pet name.

static list_pet_items(*items*)
Lists all pet-related inventory items.

Parameters **items** (*dict*) – The Habitica user.items dictionary.

notify(*message*, ***kwargs*)
Notify the Habitica user.

If this is a dry run, then the message is logged. Otherwise the message is logged and posted to the Habitica notification panel.

Parameters

- **message** (*str*) – The message.
- **panel** (*bool*) – If True, the Habitica panel is updated.

static supports_dry_runs()
The PetCare plugin supports dry runs.

Returns True

Return type bool

update()

This update method will be called once on every update cycle, with the frequency determined by the value returned from *update_interval_minutes()*.

If a plugin implements a single-shot function, then update should return *False*.

Returns: bool: True if further updates are required; False if the plugin is finished and the application should shut down.

update_interval_minutes()

Indicates the required update interval in minutes.

Returns: float: The required update interval in minutes.

Sample

A sample plugin. Doesn't do anything, but it makes a good template for new plugins. Unused methods can be deleted.

class scriptabit.plugins.sample.**Sample**

Scriptabit sample plugin.

__init__()

Initialises the plugin. Generally nothing to do here other than initialise any class attributes.

activate()

Called by the plugin framework when a plugin is activated.

deactivate()

Called by the plugin framework when a plugin is deactivated.

get_arg_parser()

Gets the argument parser containing any CLI arguments for the plugin.

Note that to avoid argument name conflicts, only long argument names should be used, and they should be prefixed with the plugin-name or unique abbreviation.

Returns: argparse.ArgParser: The *ArgParser* containing the argument definitions.

initialise(*configuration*, *habitica_service*, *data_dir*)

Initialises the plugin.

Generally, any initialisation should be done here rather than in activate or __init__.

Parameters

- **configuration** (*ArgParse.Namespace*) – The application configuration.
- **habitica_service** – the Habitica Service instance.
- **data_dir** (*str*) – A writeable directory that the plugin can use for persistent data.

static supports_dry_runs()

The Sample plugin supports dry runs.

Returns True

Return type bool

update()

This update method will be called once on every update cycle, with the frequency determined by the value returned from *update_interval_minutes()*.

If a plugin implements a single-shot function, then update should return *False*.

Returns: bool: True if further updates are required; False if the plugin is finished and the application should shut down.

update_interval_minutes()

Indicates the required update interval in minutes.

Returns: float: The required update interval in minutes.

Trello

Synchronisation of Trello cards to Habitica To-Dos.

class scriptabit.plugins.trello.trello.**Trello**

Trello card synchronisation.

__tc

TrelloClient instance

__habitica_task_service

The HabiticaTaskService instance

__task_map_file

= Task mapping data file

__data_file

Sync data file name

__data

Trello.PersistentData – Persistent sync data

class PersistentData(*filename=None*)

Data that needs to be persisted.

__init__(*filename=None*)

x.__init__(...) initializes x; see help(type(x)) for signature

save(*filename*)

Saves the persistent data

_Trello__ensure_labels_exist(*boards*)

Ensures that the Trello labels used to mark task difficulty and Habitica character attributes exist.

Parameters boards (*list*) – The list of boards that are being synchronised.

static _Trello__load_authentication_credentials(*config_file_name=u'.auth.cfg'*, *section=u'trello'*)

Loads authentication credentials from an ini-style configuration file.

Parameters

- **config_file_name** (*str*) – Basename of the configuration file.
- **section** (*str*) – Configuration file section name.

Returns: dict: the selected credentials

_Trello__load_persistent_data()

Loads the persistent data

`_Trello__notify`(*sync_stats*)
notify the user about the sync stats.

Parameters **`sync_stats`** (`TaskSync.Stats`) – Stats from the last sync.

`_Trello__parse_board_configuration`()
Parses the board configuration from the command line arguments

`_Trello__save_persistent_data`()
Saves the persistent data

`__init__`()
Initialises the plugin. Generally nothing to do here other than initialise any class attributes.

`get_arg_parser`()
Gets the argument parser containing any CLI arguments for the plugin.

Note that to avoid argument name conflicts, only long argument names should be used, and they should be prefixed with the plugin-name or unique abbreviation.

Returns: argparse.ArgParser: The *ArgParser* containing the argument definitions.

`initialise`(*configuration*, *habitica_service*, *data_dir*)
Initialises the Trello plugin.

This involves loading the board and list configuration, confirming the API key and secret, obtaining the OAuth tokens if required, and instantiating the *TrelloClient* instance.

Parameters

- **`configuration`** (*`ArgParse.Namespace`*) – The application configuration.
- **`habitica_service`** (`scriptabit.HabiticaService`) – the Habitica Service instance.
- **`data_dir`** (*`str`*) – A writeable directory that the plugin can use for persistent data.

`static supports_dry_runs`()
The Trello plugin supports dry runs.

Returns True

Return type bool

`update`()
This update method will be called once on every update cycle, with the frequency determined by the value returned from *update_interval_minutes()*.

If a plugin implements a single-shot function, then update should return *False*.

Returns: bool: True if further updates are required; False if the plugin is finished and the application should shut down.

`update_interval_minutes`()
Indicates the required update interval in minutes.

Returns: float: The required update interval in minutes.

CHAPTER 2

Indices and tables

- genindex
- modindex

Python Module Index

Index