FlexGet Documentation

Release 1.2.355.dev

FlexGet

October 08, 2015

Contents

This documentation contains developer guide, technical core documentation and relevant plugin documentation that is relevant to the developers wishing to extend FlexGet.

Indices and tables

- genindex
- modindex
- search

CHAPTER 1

Instructions for aspiring developer

1.1 Introduction

We welcome all new developers and contributors with very friendly community. Join #FlexGet @ freenode.

1.1.1 Technologies used

List of external libraries and utilities. As for new libraries, we require that all of them are installable on Windows trough `pip`. This will exclude things that require compilers like LXML.

Core

- SQLAlchemy
- BeautifulSoup
- Feedparser
- Python-Requests
- PyNBZ
- Jinja2
- PyYaml
- jsonschema
- Some smaller misc libraries

HTTPServer

- Flask
- Jinja2
- CherryPy

CherryPy is only used for WSGI server.

1.1.2 How do I get started?

Set up development environment, which is basically just two steps.

1. GIT clone our repository.
2. - Either the wheel package of setuptools and pip need to be in the directory with bootstrap.py
   - Or virtualenv package needs to be installed.
3. Run `bootstrap.py` with Python 2.6.x - 2.7.x.

For easier collaboration we recommend forking us on github and sending pull request. Once we see any semi-serious input from a developer we will grant write permissions to our central repository. You can also request this earlier if you wish.

If you are new to Git there are several interactive tutorials you can try to get you started including tryGit and LearnGitBranching.

1.1.3 Environment

Once you have bootstrapped the environment you have fully functional FlexGet in a virtual environment in your clone directory. You can easily add or modify existing plugins in here and it will not mess your other FlexGet instances in any way. The commands in the documentation expect the virtual environment to be activated. If you don't activate it you must run commands explicitly from under environment `bin` directory or `scripts` in windows. E.g. `flexget` would be `bin/flexget` (at project root) in unactivated virtual environment.

How to activate virtualenv under linux:

```
source bin/activate
```

FlexGet project uses paver to provide development related utilities and tasks. Run `paver --help` to see what commands are available. Some of these will be mentioned later.

1.1.4 Code quality

Unit tests

There are currently over 250 unit tests ensuring that existing functionality is not accidentally broken. Unit tests can be invoked with the installation of additionnal requirments:

    pip install jenkins-requirements.txt

Easiest way to run tests is trough paver:

```
paver test
```

By default no online tests are executed, these can be enabled with `--online` argument. There are other ways to run the tests as well, more specifically we use nose framework.

Run single test file via nose:

```
nosetests test_file
```

Run single test suite (class):

```
nosetests test_file:class
```

Run single test case from suite:

```
nosetests test_file:class.case
```

Live example:

```
nosetests test_seriesparser:TestSeriesParser.test_basic
```

Note: Don't use .py extension or include path with these. Configuration file `setup.cfg` defines needed parameters for Nose.

Project has Jenkins CI server which polls master branch and makes runs tests and makes new build if they pass.

Unit tests are not mandatory for a plugin to be included in the FlexGet distribution but it makes maintaining our code trough project life and refactoring so much easier.

Code Style

All code should be formatted according to Python PEP8 recommendations. With the exception of line length limit at 79 characters. FlexGet uses 120 characters instead.

To run PEP8 checker:

```
paver pep8
```

We do have some violations in our codebase, but new code should not add any.

1.2 Mock data

If you're not really hard core into TDD you still need some practical way to test how your plugin or changes behave. From here you can find some ways to achieve that.

1.2.1 Mock input

Using special input plugin called `mock` to produce almost any kind of entries in a task. This is probably one of the best ways to test things outside TDD.

Example:

yaml

tasks:

my-test:

mock:

- {title: 'title of test', description: 'foobar'}

my_custom_plugin: do_stuff: yes

This will generate one entry in the task, notice that entry has two mandatory fields `title` and `url`. If `url` is not defined the mock plugin will generate random url for localhost. The `description` filed is just arbitary field that we define in here. We can define any kind of basic text, number, list or dictionary fields in here.

1.2.2 Inject

The argument `--inject` is very useful during development, assuming previous example configuration you could try with some other title simply running following.

Example:

```
flexget --inject "another test title"
```

The `--inject` will disable any other inputs in the task. It is possible to set arbitrary fields trough inject much like with mock. See full documentation here.

1.2.3 Commandline values

The plugin cli config may be useful if you need to try bunch of different values in the configuration file. It allows placing variables in the configuration file.

Example:

yaml

task:

my-test:

mock:

- {title: foobar}

regexp:

accept:

- $regexp

Run with command:

```
flexget --cli-config "regexp=foobar"
```

1.3 TDD in practice

Simple example how to create a plugin with TDD principles.

> **Warning:** Ironically, this is an untested example :)

1.3.1 Create test

Write new test case called `tests/test_hello.py`.

```
from tests import FlexGetBase

class TestHello(FlexGetBase):

    __yaml__ = """
        tasks:
          test:
```

```
            mock:                  # let's use this plugin to create test data
              - {title: 'foobar'} # we can omit url if we do not care about it, in this case mock wi
            hello: yes             # our plugin, no relevant configuration yet ...
    """

    def test_feature(self):
      # run the task
      self.execute_task('test')
```

Try running the test with nosetests:

```
nosetests test_hello
```

It should complain that the plugin hello does not exists, that's because we haven't yet created it. Let's do that next.

1.3.2 Create plugin

Create new file called `flexget/plugins/output/hello.py`.

Within this file we will add our plugin.

```
from __future__ import unicode_literals, division, absolute_import

from flexget import plugin
from flexget.event import event


class Hello(object):
    pass

@event('plugin.register')
def register_plugin():
    plugin.register(Hello, 'hello', api_ver=2)
```

After this the unit tests should pass again. Try running them.

1.3.3 Add test

Now our example plugin will be very simple, we just want to add new field to each entry called `hello` with value `True`.

Let's supplement the testsuite with the test.

```
from tests import FlexGetBase

class TestHello(FlexGetBase):

    __yaml__ = """
        tasks:
          test:
            mock:                  # let's use this plugin to create test data
              - {title: 'foobar'} # we can omit url if we do not care about it, in this case mock wi
            hello: yes             # our plugin, no relevant configuration yet ...
    """

    def test_feature(self):
      # run the task
```

```
        self.execute_task('test')
        for entry in self.task.entries:
            self.assertEqual(entry.get('hello'), True)
```

This should fail as we do not currently have such functionality in the plugin.

1.3.4 Add functionality to plugin

Continue by implementing the test case.

```
from __future__ import unicode_literals, division, absolute_import

from flexget import plugin
from flexget.event import event


class Hello(object):
    def on_task_filter(self, task, config):
        for entry in task.entries:
            entry['hello'] = True

@event('plugin.register')
def register_plugin():
    plugin.register(Hello, 'hello', api_ver=2)
```

1.3.5 Summary

This demonstrates main principle and workflow behind TDD and shows how it can be achieved with FlexGet.

1.4 Database

FlexGet uses SQLAlchemy for database access. There are however some custom additions that developers should be aware of.

1.4.1 Migrations

The plugin system tries to make each plugin as much separated as possible. When the need for schema migrations became too overwhelming the team evaluated few possibilities but none of them were able to version each plugin separately. Even the latest official tool from SQLAlchemy authors does not seem to make it easily possible.

Because this special requirement we had to make custom implementation for migrations.

Migrate Base

The plugin needs only to use custom `Base` from `flexget.schema.versioned_base`.

```
SCHEMA_VER = 0
Base = schema.versioned_base('plugin_name_here', SCHEMA_VER)
```

This will automatically track all the declarative models plugin uses. When there are changes in the plugin database, increment the number and add migration code. This is done with decorated function in the following manner.

```
@schema.upgrade('series')
def upgrade(ver, session):
    if ver==1:
        # upgrade actions
        ver = 2
    return ver
```

Warning: The upgrade function can NOT use any declarative models. That will break sooner or later when models are evolving.

There are several helper functions available in *flexget.utils.sqlalchemy_utils* that allow querying database reflectively. Use SQLAlchemy's core expression to make alterations to the table structure.

1.4.2 Database cleanups

If the plugin accumulates data into database that should be cleaned at some point *manager.db_cleanup* event should be used. This will be automatically called every 7 days in non intrusive way.

```
@event('manager.db_cleanup')
def db_cleanup(session):
    # cleanup actions here
```

1.5 Plugin Schemas

Plugins define their desired form of their config using draft 4 of the JSON Schema specification. The schema for a plugin should be stored `schema` attribute of the plugin class. The schema is used for several things including:

- Validating the config format matches what is expected
- Set defaults in the config that user did not provide
- Generating a form for the webui config editor

You can run the *test_config_schema.py* test in the suite to test the validity of your plugin's schema. The error messages it produces may help you fix your schema if you are having trouble. Note that this doesn't check the schema validates what you want, just that it is a valid json schema.

The following list of keywords is not exhaustive, just a general primer, as well as some FlexGet specific notes. The JSON schema spec should be referred to for more details, or if a keyword is not covered here. Take not that our schemas will be defined as python objects equivalent to parsed JSON. The full list of valid keywords can be found in section 5 of the validaton spec.

1.5.1 Keywords

type

The `type` keyword specifies what primitive data type a given item in the config should have. It can either be a single type, or a list of types. The `type` keyword should be specified in almost every schema, even when other keywords are included which might make it redundant, as it is used to select which branch should be chosen to show errors when the config can take multiple forms. The types in JSON schema are slightly different than their python counterparts. Here are the valid options, along with the python types they map to:

JSON Schema type	Python type
string	unicode
boolean	bool
number	float
integer	int
array	list
object	dict
null	type(None)

items

This keyword is used to validate the content of a list ('array' in json schema terms.) Its value should be a schema that each item in the list matches.

The following example describes a list of strings:

```
{"type": "array", "items": {"type": "string"}}
```

properties

This keyword is used to validate the values in a dictionary. It should be a dictionary mapping from key name, to schema which validates the value for that key.

The following example describes a dictionary with two keys, 'a', and 'b', both of which must be integers (`additionalProperties` will be explained below):

```
{
    "type": "object",
    "properties": {
        "a": {"type": "integer"},
        "b": {"type": "integer"}
    },
    "additionalProperties": False
}
```

additionalProperties

By default, JSON schema will allow any keys which are not defined in the `properties` dictionary without validation. To disallow extra keys, use the `{"additionalProperties": False}` form, as in the above example. This should be used in almost every schema which defines the `properties` keyword. The other use for this keyword is if you want to allow a dictionary with any keys, but still require the values to match a schema.

The following example allows a dictionary with any keys, as long as the values are strings:

```
{
    "type": "object",
    "additionalProperties": {"type": "string"}
}
```

oneOf and anyOf

These keywords are used when the config could take more than one format. The value should be a list of schemas one of which, or any of which must match, depending on the keyword used.

The following schema will allow either a boolean or an integer:

```
{"oneOf": [{"type": "boolean"}, {"type": "integer"}]}
```

format

The format keyword is used to make sure a string follows a specific format. Here are the format validators included with FlexGet, along with what they validate:

email email addresses

quality FlexGet quality, e.g. `702p hdtv`

quality_requirements FlexGet quality requirements specifier, e.g. `720p-1080p hdtv+`

interval A text representation of a time interval, e.g. `3 hours`, `10 minutes` Intervals in this format can be parsed to a `datetime.timedelta` object using the utility function *`flexget.utils.tools.parse_timedelta()`*

regex valid regular expression

file an existing file on the local filesystem

path an existing directory on the local filesystem (if path contains Jinja, only validates path exists before first Jinja component of path)

The following schema checks for valid regex:

```
{"type": "string", "format": "regex"}
```

$ref

This keyword is used to reference a schema defined somewhere else. The most common use of this keyword will be to allow a plugin to take other plugins within their configuration. It takes the form of an URI reference. The fragment part should be a JSON pointer to a section of the referenced document. If *only* a fragment portion of an URI is specified, the base document is assumed to be the current schema.

The following schema allows a dictionary with keys equal to plugin names (which have input phase handlers,) and values equal to the configuration required for that plugin. We don't actually define the validation keywords here, we are just referencing an already built schema which has been registered by some other plugin or component of FlexGet:

```
{"$ref": "/schema/plugins?phase=input"}
```

definitions

This keyword does not affect validation, it is merely used to define parts of your schema that may get re-used in more than one place. It should be in the form of a dictionary mapping arbitrary names to a schema.

The following schema defines a definition called posNumber, and references it from two places within the schema:

```
{
    "type": "object",
    "properties": {
        "numberA": {"$ref": "#/definitions/posNumber"},
        "numberB": {"$ref": "#/definitions/posNumber"}
    },
    "additionalProperties": False,
    "definitions": {
```

```
        "posNumber": {"type": "number", "minimum": 0}
    }
}
```

The `$ref` used in this example included a fragment part of an URI only, so it references this schema, and drills down into it with a JSON pointer.

`title` and `description`

The `title` and `description` keywords are not used during validation at all. If provided, they will be used to display more information to the user in the configuration editor.

`default`

The `default` keyword is not used during validation either. It will be used to fill in default values for properties in the config that the user has not provided. This will be done automatically before the parsed config is passed to the plugin.

CHAPTER 2

Core documentation

Describe FlexGet framework.

2.1 flexget Package

2.1.1 flexget Package

flexget.__init__.**main**(*args=None*)
Main entry point for Command Line Interface

2.1.2 api Module

2.1.3 config_schema Module

class flexget.config_schema.**RefResolver**(**args*, ***kwargs*)
Bases: jsonschema.validators.RefResolver

flexget.config_schema.**get_schema**()

flexget.config_schema.**is_file**(*instance*)

flexget.config_schema.**is_interval**(*interval_string*)

flexget.config_schema.**is_path**(*instance*)

flexget.config_schema.**is_percent**(*percent_string*)

flexget.config_schema.**is_quality**(*instance*)

flexget.config_schema.**is_quality_req**(*instance*)

flexget.config_schema.**is_regex**(*instance*)

flexget.config_schema.**is_size**(*size_string*)

flexget.config_schema.**is_time**(*time_string*)

flexget.config_schema.**is_url**(*instance*)

flexget.config_schema.**one_or_more**(*schema*)
Helper function to construct a schema that validates items matching *schema* or an array containing items matching *schema*.

flexget.config_schema.**parse_interval**(*interval_string*)
Takes an interval string from the config and turns it into a datetime.timedelta object.

flexget.config_schema.**parse_percent**(*percent_input*)
Takes a size string from the config and turns it into int(bytes).

flexget.config_schema.**parse_size**(*size_input*)
Takes a size string from the config and turns it into int(bytes).

flexget.config_schema.**parse_time**(*time_string*)
Parse a time string from the config into a datetime.time object.

flexget.config_schema.**process_config**(*config*, *schema=None*, *set_defaults=True*)
Validates the config, and sets defaults within it if *set_defaults* is set. If schema is not given, uses the root config schema.

Returns A list with :class:‘jsonschema.ValidationError‘s if any

flexget.config_schema.**register_config_key**(*key*, *schema*, *required=False*)
Registers a valid root level key for the config.

Parameters

- **key** (*string*) – Name of the root level key being registered.
- **schema** (*dict*) – Schema for the key.
- **required** (*bool*) – Specify whether this is a mandatory key.

flexget.config_schema.**register_schema**(*path*, *schema*)
Register *schema* to be available at *path* for $refs

Parameters

- **path** – Path to make schema available
- **schema** – The schema, or function which returns the schema

flexget.config_schema.**resolve_ref**(*uri*)
Finds and returns a schema pointed to by *uri* that has been registered in the register_schema function.

flexget.config_schema.**select_child_errors**(*validator*, *errors*)
Looks through subschema errors, if any subschema is determined to be the intended one, (based on ‘type’ keyword errors,) errors from its branch will be released instead of the parent error.

flexget.config_schema.**set_error_message**(*error*)
Create user facing error message from a jsonschema.ValidationError *error*

flexget.config_schema.**validate_anyOf**(*validator*, *anyOf*, *instance*, *schema*)

flexget.config_schema.**validate_oneOf**(*validator*, *oneOf*, *instance*, *schema*)

flexget.config_schema.**validate_properties_w_defaults**(*validator*, *properties*, *instance*, *schema*)

2.1.4 db_schema Module

class flexget.db_schema.**Meta**
Bases: type

Metaclass for objects returned by versioned_base factory

class flexget.db_schema.**PluginSchema**(*plugin*, *version=0*)
Bases: sqlalchemy.ext.declarative.api.Base

id

plugin

version

exception flexget.db_schema.**UpgradeImpossible**
Bases: exceptions.Exception

Exception to be thrown during a db upgrade function which will cause the old tables to be removed and recreated from the new model.

flexget.db_schema.**after_table_create**(*event*, *target*, *bind*, *tables=None*, ***kw*)
Sets the schema version to most recent for a plugin when it's tables are freshly created.

flexget.db_schema.**register_plugin_table**(*tablename*, *plugin*, *version*)

flexget.db_schema.**upgrade**(*plugin*)
Used as a decorator to register a schema upgrade function.

The wrapped function will be passed the current schema version and a session object. The function should return the new version of the schema after the upgrade.

There is no need to commit the session, it will commit automatically if an upgraded schema version is returned.

Example:

```
from flexget import schema
@schema.upgrade('your_plugin')
def upgrade(ver, session):
    if ver == 2:
        # upgrade
        ver = 3
    return ver
```

flexget.db_schema.**versioned_base**(*plugin*, *version*)
Returns a class which can be used like Base, but automatically stores schema version when tables are created.

2.1.5 entry Module

class flexget.entry.**Entry**(**args*, ***kwargs*)
Bases: flexget.utils.lazy_dict.LazyDict

Represents one item in task. Must have *url* and *title* fields.

Stores automatically *original_url* key, which is necessary because plugins (eg. urlrewriters) may change *url* into something else and otherwise that information would be lost.

Entry will also transparently convert all ascii strings into unicode and raises *EntryUnicodeError* if conversion fails on any value being set. Such failures are caught by *Task* and trigger *abort()*.

accept(*reason=None*, ***kwargs*)

accepted

add_hook(*action*, *func*, ***kwargs*)
Add a hook for action to this entry.

Parameters

- **action** (*string*) – One of: 'accept', 'reject', 'fail', 'complete'
- **func** – Function to execute when event occurs

- **kwargs** – Keyword arguments that should be passed to `func`

Raises ValueError when given an invalid `action`

complete(***kwargs*)

fail(*reason=None*, ***kwargs*)

failed

isvalid()

Returns True if entry is valid. Return False if this cannot be used.

Return type bool

on_accept(*func*, ***kwargs*)
Register a function to be called when this entry is accepted.

Parameters

- **func** – The function to call
- **kwargs** – Keyword arguments that should be passed to the registered function

on_complete(*func*, ***kwargs*)
Register a function to be called when a `Task` has finished processing this entry.

Parameters

- **func** – The function to call
- **kwargs** – Keyword arguments that should be passed to the registered function

on_fail(*func*, ***kwargs*)
Register a function to be called when this entry is failed.

Parameters

- **func** – The function to call
- **kwargs** – Keyword arguments that should be passed to the registered function

on_reject(*func*, ***kwargs*)
Register a function to be called when this entry is rejected.

Parameters

- **func** – The function to call
- **kwargs** – Keyword arguments that should be passed to the registered function

reject(*reason=None*, ***kwargs*)

rejected

render(*template*)
Renders a template string based on fields in the entry.

Parameters **template** (*string*) – A template string that uses jinja2 or python string replacement format.

Returns The result of the rendering.

Return type string

Raises RenderError If there is a problem.

run_hooks(*action*, ***kwargs*)
Run hooks that have been registered for given `action`.

Parameters

- **action** – Name of action to run hooks for
- **kwargs** – Keyword arguments that should be passed to the registered functions

safe_str()

take_snapshot(*name*)
Takes a snapshot of the entry under *name*. Snapshots can be accessed via `snapshots`. :param string name: Snapshot name

trace(*message*, *operation=None*, *plugin=None*)
Adds trace message to the entry which should contain useful information about why plugin did not operate on entry. Accept and Reject messages are added to trace automatically.

Parameters

- **message** (*string*) – Message to add into entry trace.
- **operation** (*string*) – None, reject, accept or fail
- **plugin** – Uses task.current_plugin by default, pass value to override

undecided

update_using_map(*field_map*, *source_item*, *ignore_none=False*)
Populates entry fields from a source object using a dictionary that maps from entry field names to attributes (or keys) in the source object.

Parameters

- **field_map** (*dict*) – A dictionary mapping entry field names to the attribute in source_item (or keys, if source_item is a dict)(nested attributes/dicts are also supported, separated by a dot,) or a function that takes source_item as an argument
- **source_item** – Source of information to be used by the map
- **ignore_none** – Ignore any None values, do not record it to the Entry

exception `flexget.entry.`**EntryUnicodeError**(*key*, *value*)
Bases: `exceptions.Exception`

This exception is thrown when trying to set non-unicode compatible field value to entry.

2.1.6 `event` Module

Provides small event framework

class `flexget.event.`**Event**(*name*, *func*, *priority=128*)
Bases: `object`

Represents one registered event.

`flexget.event.`**add_event_handler**(*name*, *func*, *priority=128*)

Parameters

- **name** (*string*) – Event name
- **func** (*function*) – Function that acts as event handler
- **priority** – Priority for this hook

Returns Event created

Return type *Event*

Raises Exception If *func* is already registered in an event

flexget.event.**event**(*name*, *priority=128*)
Register event to function with a decorator

flexget.event.**fire_event**(*name*, **args*, ***kwargs*)
Trigger an event with *name*. If event is not hooked by anything nothing happens. If a function that hooks an event returns a value, it will replace the first argument when calling next function.

Parameters

- **name** – Name of event to be called
- **args** – List of arguments passed to handler function
- **kwargs** – Key Value arguments passed to handler function

flexget.event.**get_events**(*name*)

Parameters **name** (*String*) – event name

Returns List of *Event* for *name* ordered by priority

flexget.event.**remove_event_handler**(*name*, *func*)
Remove *func* from the handlers for event *name*.

flexget.event.**remove_event_handlers**(*name*)
Removes all handlers for given event *name*.

2.1.7 ipc Module

class flexget.ipc.**ClientService**(*conn*)
Bases: rpyc.core.service.Service

exposed_console(*text*)

exposed_version()

on_connect()
Make sure the client version matches our own.

class flexget.ipc.**DaemonService**(*conn*)
Bases: rpyc.core.service.Service

client_console(*text*)

client_out_stream

exposed_handle_cli(*args*)

exposed_version()

manager = None

class flexget.ipc.**IPCClient**(*port*, *password*)
Bases: object

close()

class flexget.ipc.**IPCServer**(*manager*, *port=None*)
Bases: threading.Thread

authenticator(*sock*)

run()

shutdown()

class flexget.ipc.**RemoteStream**(*writer*)
Bases: object

Used as a filelike to stream text to remote client. If client disconnects while this is in use, an error will be logged, but no exception raised.

write(*data*)

2.1.8 logger Module

class flexget.logger.**FlexGetFormatter**
Bases: logging.Formatter

Custom formatter that can handle both regular log records and those created by FlexGetLogger

flexget_fmt = u'%(asctime)-15s %(levelname)-8s %(name)-13s %(task)-15s %(message)s'

format(*record*)

class flexget.logger.**FlexGetJsonFormatter**
Bases: logging.Formatter

fields = [u'asctime', u'levelname', u'name', u'task', u'task_id', u'message']

format(*record*)

class flexget.logger.**FlexGetLogger**(*name*, *level=0*)
Bases: logging.Logger

Custom logger that adds trace and verbose logging methods, and contextual information to log records.

makeRecord(*name*, *level*, *fn*, *lno*, *msg*, *args*, *exc_info*, *func=None*, *extra=None*)

trace(*msg*, **args*, ***kwargs*)
Log at TRACE level (more detailed than DEBUG).

verbose(*msg*, **args*, ***kwargs*)
Log at VERBOSE level (displayed when FlexGet is run interactively.)

class flexget.logger.**RollingBuffer**
Bases: collections.deque

File-like that keeps a certain number of lines of text in memory.

write(*line*)

class flexget.logger.**SessionFilter**(*session_id*)
Bases: logging.Filter

filter(*record*)

flexget.logger.**capture_output**(**args*, ***kwds*)
Context manager which captures all log and console output to given *stream* while in scope.

flexget.logger.**console**(*text*)
Print to console safely. Output is able to be captured by different streams in different contexts.

Any plugin wishing to output to the user's console should use this function instead of print so that output can be redirected when FlexGet is invoked from another process.

flexget.logger.**get_capture_loglevel**()
If output is currently being redirected to a stream, returns declared loglevel for that stream.

flexget.logger.**get_capture_stream**()
If output is currently being redirected to a stream, returns that stream.

flexget.logger.**get_level_no**(*level*)

flexget.logger.**initialize**(*unit_test=False*)
Prepare logging.

flexget.logger.**start**(*filename=None*, *filename_json=None*, *level=20*, *to_console=True*, *to_file=True*)
After initialization, start file logging.

flexget.logger.**task_logging**(**args*, ***kwds*)
Context manager which adds task information to log messages.

2.1.9 manager Module

class flexget.manager.**Manager**(*args*)
Bases: object

Manager class for FlexGet

Fires events:

- manager.initialize

  The first time the manager is initialized, before config is loaded

- manager.before_config_load

  Before the config file is loaded from disk

- manager.before_config_validate

  When updating the config, before the validator is run on it

- manager.config_updated

  After a configuration file has been loaded or changed (and validated) this event is fired

- manager.startup

  After manager has been initialized. This is when application becomes ready to use, however no database lock is present, so the database must not be modified on this event.

- manager.lock_acquired

  The manager does not always require a lock on startup, if one is requested, this event will run when it has been acquired successfully

- manager.upgrade

  If any plugins have declared a newer schema version than exists in the database, this event will be fired to allow plugins to upgrade their tables

- manager.shutdown_requested

  When shutdown has been requested. Any plugins which might add to execution queue should stop when this is fired.

- manager.shutdown

  When the manager is exiting

•manager.execute.completed

If execution in current process was completed

•manager.daemon.started

•manager.daemon.completed

•manager.db_cleanup

acquire_lock(**args*, ***kwds*)

Parameters **event** (*bool*) – If True, the 'manager.lock_acquired' event will be fired after a lock is obtained

check_ipc_info()
If a daemon has a lock on the database, return info to connect to IPC.

check_lock()
Returns True if there is a lock on the database.

config_changed()
Makes sure that all tasks will have the config_modified flag come out true on the next run. Useful when changing the db and all tasks need to be completely reprocessed.

crash_report()
This should be called when handling an unexpected exception. Will create a new log file containing the last 50 debug messages as well as the crash traceback.

daemon_command(*options*)
Handles the 'daemon' CLI command.

Fires events:

•manager.daemon.started

•manager.daemon.completed

Parameters **options** – argparse options

daemonize()
Daemonizes the current process. Returns the new pid

db_cleanup(*force=False*)
Perform database cleanup if cleanup interval has been met.

Fires events:

•manager.db_cleanup

If interval was met. Gives session to do the cleanup as a parameter.

Parameters **force** (*bool*) – Run the cleanup no matter whether the interval has been met.

execute(*options=None*, *output=None*, *loglevel=None*, *priority=1*)
Run all (can be limited with options) tasks from the config.

Parameters

- **options** – Either an `argparse.Namespace` instance, or a dict, containing options for execution
- **output** – If a file-like object is specified here, log messages and stdout from the execution will be written to it.

- **priority** – If there are other executions waiting to be run, they will be run in priority order, lowest first.

Returns a list of `threading.Event` instances which will be set when each respective task has finished running

execute_command(*options*)
Handles the 'execute' CLI command.

If there is already a task queue running in this process, adds the execution to the queue. If FlexGet is being invoked with this command, starts up a task queue and runs the execution.

Fires events:

•manager.execute.started

•manager.execute.completed

Parameters **options** – argparse options

find_config(*create=False*)
Find the configuration file.

Parameters **create** (*bool*) – If a config file is not found, and create is True, one will be created in the home folder

Raises *IOError* when no config file could be found, and *create* is False.

handle_cli(*options=None*)
Dispatch a cli command to the appropriate function.

•*execute_command()*

•*daemon_command()*

•CLI plugin callback function

The manager should have a lock and be initialized before calling this method.

Parameters **options** – argparse options for command. Defaults to options that manager was instantiated with.

has_lock

init_sqlalchemy()
Initialize SQLAlchemy

initialize()
Load plugins, database, and config. Also initializes (but does not start) the task queue and ipc server. This should only be called after obtaining a lock.

load_config()
Loads the config file from disk, validates and activates it.

Raises *ValueError* if there is a problem loading the config file

options = None

release_lock()

save_config()
Dumps current config to yaml config file

setup_yaml()
Sets up the yaml loader to return unicode objects for strings by default

shutdown(*finish_queue=True*)
Request manager shutdown.

Parameters **finish_queue** (*bool*) – Should scheduler finish the task queue

start()
Starting point when executing from commandline, dispatch execution to correct destination.

If there is a FlexGet process with an ipc server already running, the command will be sent there for execution and results will be streamed back. If not, this will attempt to obtain a lock, initialize the manager, and run the command here.

tasks
A list of tasks in the config

unit_test = False

update_config(*config*)
Provide a new config for the manager to use.

Raises *ValueError* and rolls back to previous config if the provided config is not valid.

validate_config(*config=None*)
Check all root level keywords are valid. Config may be modified by before_config_validate hooks. Modified config will be returned.

Parameters **config** – Config to check. If not provided, current manager config will be checked.

Raises *ValueError* when config fails validation. There will be an *errors* attribute with the schema errors.

Returns Final validated config.

write_lock(*ipc_info=None*)

2.1.10 options Module

class flexget.options.**ArgumentParser**(***kwargs*)
Bases: argparse.ArgumentParser

Mimics the default argparse.ArgumentParser class, with a few distinctions, mostly to ease subparser usage:

- If *add_subparsers* is called with the *nested_namespaces* kwarg, all subcommand options will be stored in a nested namespace based on the command name for the subparser
- Adds the *add_subparser* method. After *add_subparsers* has been called, the *add_subparser* method can be used instead of the *add_parser* method of the object returned by the *add_subparsers* call.
- *add_subparser* takes takes the *parent_defaults* argument, which will set/change the defaults for the parent parser when that subparser is selected.
- The *get_subparser* method will get the *ArgumentParser* instance for an existing subparser on this parser
- For any arguments defined both in this parser and one of its subparsers, the selected subparser default will override the main one.
- Adds the *set_post_defaults* method. This works like the normal argparse *set_defaults* method, but all actions and subparsers will be run before any of these defaults are set.

- Command shortening: If the command for a subparser is abbreviated unambiguously, it will still be accepted.
- The add_argument *nargs* keyword argument supports a range of arguments, e.g. ‘“2-4”
- If the *raise_errors* keyword argument to *parse_args* is True, a *ParserError* will be raised instead of sys.exit
- If the *file* argument is given to *parse_args*, output will be printed there instead of sys.stdout or stderr

add_argument(**args*, ***kwargs*)

add_subparser(*name*, ***kwargs*)
Adds a parser for a new subcommand and returns it.

Parameters

- **name** – Name of the subcommand
- **parent_defaults** – Default argument values which should be supplied to the parent parser if this subparser is selected.

add_subparsers(***kwargs*)

Parameters **nested_namespaces** – If True, options from subparsers will appear in nested namespace under the subparser name.

error(*msg*)

file = None

get_subparser(*name*, *default=<object object>*)

parse_args(*args=None*, *namespace=None*, *raise_errors=False*, *file=None*)

Parameters **raise_errors** – If this is true, errors will be raised as ‘ParserError‘s instead of calling sys.exit

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

set_post_defaults(***kwargs*)
Like set_defaults method, but these defaults will be defined after parsing instead of before.

class flexget.options.**CoreArgumentParser**(***kwargs*)
Bases: *flexget.options.ArgumentParser*

The core argument parser, contains the manager arguments, command parsers, and plugin arguments.

Warning: Only gets plugin arguments if instantiated after plugins have been loaded.

add_subparsers(***kwargs*)

parse_args(**args*, ***kwargs*)

class flexget.options.**CronAction**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)
Bases: argparse.Action

class flexget.options.**DebugAction**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)
Bases: argparse.Action

class flexget.options.**DebugTraceAction**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)
Bases: argparse.Action

class flexget.options.**InjectAction**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)
Bases: argparse.Action

class flexget.options.**NestedSubparserAction**(**args*, ***kwargs*)
Bases: argparse._SubParsersAction

add_parser(*name*, *parent_defaults=None*, ***kwargs*)

class flexget.options.**ParseExtrasAction**(*option_strings*, *parser*, *help=None*, *metavar=None*, *dest=None*, *required=False*)
Bases: argparse.Action

This action will take extra arguments, and parser them with a different parser.

exception flexget.options.**ParserError**(*message*, *parser*)
Bases: exceptions.Exception

class flexget.options.**ScopedNamespace**(***kwargs*)
Bases: argparse.Namespace

class flexget.options.**VersionAction**(*option_strings*, *version=None*, *dest='==SUPPRESS=='*, *default='==SUPPRESS=='*, *help="show program's version number and exit"*)
Bases: argparse._VersionAction

Action to print the current version. Also checks latest release revision.

flexget.options.**get_parser**(*command=None*)

flexget.options.**register_command**(*command*, *callback*, ***kwargs*)
Register a callback function to be executed when flexget is launched with the given *command*.

Parameters

- **command** – The command being defined.
- **callback** – Callback function executed when this command is invoked from the CLI. Should take manager instance and parsed argparse namespace as parameters.
- **kwargs** – Other keyword arguments will be passed to the arparse.ArgumentParser constructor

Returns An argparse.ArgumentParser instance ready to be configured with the options for this command.

flexget.options.**required_length**(*nmin*, *nmax*)
Generates a custom Action to validate an arbitrary range of arguments.

flexget.options.**unicode_argv**()
Like sys.argv, but decodes all arguments.

2.1.11 plugin Module

Plugin Loading & Management.

exception flexget.plugin.**PluginWarning**(*value*, *logger=<flexget.logger.FlexGetLogger object>*, ***kwargs*)
Bases: exceptions.Warning

exception flexget.plugin.**PluginError**(*value*, *logger=<flexget.logger.FlexGetLogger object>*, ***kwargs*)
Bases: exceptions.Exception

flexget.plugin.**register_task_phase**(*name*, *before=None*, *after=None*)
Adds a new task phase to the available phases.

flexget.plugin.**get_plugin_by_name**(*name*, *issued_by=u'???'*)
Get plugin by name, preferred way since this structure may be changed at some point.

flexget.plugin.**get_plugins_by_group**(*group*)
Deprecated since version 1.0.3328: Use `get_plugins()` instead

Return an iterator over all plugins with in specified group.

flexget.plugin.**get_plugin_keywords**()
Return iterator over all plugin keywords.

flexget.plugin.**get_plugins_by_phase**(*phase*)
Deprecated since version 1.0.3328: Use `get_plugins()` instead

Return an iterator over all plugins that hook :phase:

flexget.plugin.**get_phases_by_plugin**(*name*)
Return all phases plugin :name: hooks

class flexget.plugin.**internet**(*logger=None*)
Bases: `object`

@internet decorator for plugin phase methods.

Catches all internet related exceptions and raises PluginError with relevant message. Task handles PluginErrors by aborting the task.

flexget.plugin.**priority**(*value*)
Priority decorator for phase methods

2.1.12 `task` Module

class flexget.task.**EntryContainer**(*iterable=None*)
Bases: `list`

Container for a list of entries, also contains accepted, rejected failed iterators over them.

accepted

entries

failed

rejected

undecided

class flexget.task.**EntryIterator**(*entries*, *states*)
Bases: `object`

An iterator over a subset of entries to emulate old task.accepted/rejected/failed/entries properties.

reverse()

sort(**args*, ***kwargs*)

class flexget.task.**Task**(*manager*, *name*, *config=None*, *options=None*, *output=None*, *loglevel=None*, *priority=None*)
Bases: `object`

Represents one task in the configuration.

Fires events:

•task.execute.before_plugin

Before a plugin is about to be executed. Note that since this will also include all builtin plugins the amount of calls can be quite high

`parameters: task, keyword`

•task.execute.after_plugin

After a plugin has been executed.

`parameters: task, keyword`

•task.execute.started

Before a task starts execution

•task.execute.completed

After task execution has been completed

`parameters: task`

abort(*reason=u'Unknown'*, *silent=False*)
Abort this task execution, no more plugins will be executed except the abort handling ones.

accepted
Deprecated since version Use: API v3

all_entries
Deprecated since version Use: API v3

config_changed()
Sets config_modified flag to True for the remainder of this run. Used when the db changes, and all entries need to be reprocessed.

copy()

disable_phase(*phase*)
Disable `phase` from execution.

All disabled phases are re-enabled by `Task._reset()` after task execution has been completed.

Parameters **phase** (*string*) – Name of `phase`

Raises ValueError *phase* could not be found.

entries
Deprecated since version Use: API v3

execute(**args*, ***kw*)
Executes the the task.

If `enabled` is False task is not executed. Certain *`options`* affect how execution is handled.

•`options.disable_phases` is a list of phases that are not enabled for this execution.

•`options.inject` is a list of `Entry` instances used instead of running input phase.

failed
Deprecated since version Use: API v3

find_entry(*category=u'entries'*, ***values*)
Find and return *`Entry`* with given attributes from task or None

Parameters

- **category** (*string*) – entries, accepted, rejected or failed. Defaults to entries.
- **values** – Key values of entries to be searched

Returns Entry or None

is_rerun

max_reruns = 5

plugins(*phase=None*)
Get currently enabled plugins.

Parameters **phase** (*string*) – Optional, limits to plugins currently configured on given phase, sorted in phase order.

Returns An iterator over configured `flexget.plugin.PluginInfo` instances enabled on this task.

rejected
Deprecated since version Use: API v3

rerun()
Immediately re-run the task after execute has completed, task can be re-run up to *max_reruns* times.

undecided
Deprecated since version Use: API v3

static **validate_config**(*config*)

exception flexget.task.**TaskAbort**(*reason*, *silent=False*)
Bases: exceptions.Exception

class flexget.task.**TaskConfigHash**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Stores the config hash for tasks so that we can tell if the config has changed since last run.

hash

id

task

flexget.task.**register_config_key**()

flexget.task.**use_task_logging**(*func*)

2.1.13 task_queue Module

class flexget.task_queue.**TaskInfo**(*task*)
Bases: object

aborted(*reason*)

finish(*entries*)

running

start()

class flexget.task_queue.**TaskQueue**
Bases: object

Task processing thread. Only executes one task at a time, if more are requested they are queued up and run in turn.

is_alive()

put(*task*)
Adds a task to be executed to the queue.

run()

shutdown(*finish_queue=True*)
Request shutdown.

Parameters **finish_queue** (*bool*) – Should all tasks be finished before ending thread.

start()

wait()
Waits for the thread to exit. Allows abortion of task queue with ctrl-c

2.1.14 webserver Module

flexget.webserver.**register_app**(*path*, *application*)

flexget.webserver.**register_config**()

flexget.webserver.**register_home**(*route*)
Registers UI home page

flexget.webserver.**setup_server**(*manager*)
Sets up and starts/restarts the web service.

flexget.webserver.**start_page**()
Redirect user to registered UI home

flexget.webserver.**stop_server**(*manager*)
Sets up and starts/restarts the webui.

2.1.15 validator Module

Deprecated since version 1.1: Use config_schema instead

class flexget.validator.**AnyValidator**(*parent=None*, *message=None*, ***kwargs*)
Bases: *flexget.validator.Validator*

accept(*value*, ***kwargs*)

name = u'any'

class flexget.validator.**BooleanValidator**(*parent=None*, *message=None*, ***kwargs*)
Bases: *flexget.validator.Validator*

accept(*name*, ***kwargs*)

name = u'boolean'

class flexget.validator.**ChoiceValidator**(*parent=None*, ***kwargs*)
Bases: *flexget.validator.Validator*

accept(*value*, *ignore_case=False*)

Parameters

- **value** – accepted text, int or boolean
- **ignore_case** (*bool*) – Whether case matters for text values

accept_choices (*values*, ***kwargs*)
Same as accept but with multiple values (list)

name = u'choice'

class flexget.validator.**DecimalValidator** (*parent=None*, *message=None*, ***kwargs*)
Bases: *flexget.validator.Validator*

accept (*name*, ***kwargs*)

name = u'decimal'

class flexget.validator.**DictValidator** (*parent=None*, ***kwargs*)
Bases: *flexget.validator.Validator*

accept (*value*, *key=None*, *required=False*, ***kwargs*)

Parameters

- **value** – validator name, instance or function that returns an instance, which validates the given *key*
- **key** (*string*) – The dictionary key to accept
- **required** (*bool*) – = Mark this *key* as required

Raises ValueError *key* was not specified

accept_any_key (*value*, ***kwargs*)
Accepts any leftover keys in dictionary, which will be validated with *value*

accept_valid_keys (*value*, *key_type=None*, *key_validator=None*, ***kwargs*)
Accepts keys that pass a given validator, and validates them using validator specified in *value*

Parameters

- **value** – Validator name, instance or function returning an instance that will be used to validate dict values.
- **key_type** – Name of validator or list of names that determine which keys in this dict *value* will govern
- **key_validator** (Validator) – A validator instance that will be used to determine which keys in the dict *value* will govern

Raises ValueError If both *key_type* and *key_validator* are specified.

name = u'dict'

reject_key (*key*, *message=None*)
Rejects a key

reject_keys (*keys*, *message=None*)
Reject list of keys

require_key (*key*)
Flag key as mandatory

class flexget.validator.**EqualsValidator** (*parent=None*, *message=None*, ***kwargs*)
Bases: *flexget.validator.Validator*

accept (*value*, ***kwargs*)

name = u'equals'

class flexget.validator.**Errors**
Bases: object

Create and hold validator error messages.

add(*msg*)
Add new error message to current path.

back_out_errors(*num=1*)
Remove last num errors from list

count()
Return number of errors.

path_add_level(*value=u'?'*)
Adds level into error message path

path_remove_level()
Removes level from path by depth number

path_update_value(*value*)
Updates path level value

class flexget.validator.**FileValidator**(*parent=None*, *message=None*, ***kwargs*)
Bases: *flexget.validator.TextValidator*

name = u'file'

validate(*data*)

class flexget.validator.**IntegerValidator**(*parent=None*, *message=None*, ***kwargs*)
Bases: *flexget.validator.Validator*

accept(*name*, ***kwargs*)

name = u'integer'

class flexget.validator.**IntervalValidator**(*parent=None*, ***kwargs*)
Bases: *flexget.validator.RegexpMatchValidator*

name = u'interval'

class flexget.validator.**ListValidator**(*parent=None*, *message=None*, ***kwargs*)
Bases: *flexget.validator.Validator*

accept(*value*, ***kwargs*)

name = u'list'

class flexget.validator.**NumberValidator**(*parent=None*, *message=None*, ***kwargs*)
Bases: *flexget.validator.Validator*

accept(*name*, ***kwargs*)

name = u'number'

class flexget.validator.**PathValidator**(*parent=None*, *allow_replacement=False*, *allow_missing=False*, ***kwargs*)
Bases: *flexget.validator.TextValidator*

name = u'path'

class flexget.validator.**QualityRequirementsValidator**(*parent=None*, *message=None*, ***kwargs*)
Bases: *flexget.validator.TextValidator*

name = u'quality_requirements'

class flexget.validator.**QualityValidator**(*parent=None*, *message=None*, ***kwargs*)
Bases: *flexget.validator.TextValidator*

name = u'quality'

class flexget.validator.**RegexpMatchValidator**(*parent=None*, ***kwargs*)
Bases: *flexget.validator.Validator*

accept(*regexp*, ***kwargs*)

add_regexp(*regexp_list*, *regexp*)

name = u'regexp_match'

reject(*regexp*)

class flexget.validator.**RegexpValidator**(*parent=None*, *message=None*, ***kwargs*)
Bases: *flexget.validator.Validator*

accept(*name*, ***kwargs*)

name = u'regexp'

class flexget.validator.**RootValidator**(*parent=None*, *message=None*, ***kwargs*)
Bases: *flexget.validator.Validator*

accept(*value*, ***kwargs*)

name = u'root'

class flexget.validator.**TextValidator**(*parent=None*, *message=None*, ***kwargs*)
Bases: *flexget.validator.Validator*

accept(*name*, ***kwargs*)

name = u'text'

class flexget.validator.**UrlValidator**(*parent=None*, *protocols=None*, ***kwargs*)
Bases: *flexget.validator.TextValidator*

name = u'url'

class flexget.validator.**Validator**(*parent=None*, *message=None*, ***kwargs*)
Bases: object

accept(*value*, ***kwargs*)

add_parent(*parent*)

add_root_parent()

errors
Recursively return the Errors class from the root of the validator tree.

get_validator(*value*, ***kwargs*)
Returns a child validator of this one.

Parameters

- **value** – Can be a validator type string, an already created Validator instance, or a function that returns a validator instance.
- **kwargs** – Keyword arguments are passed on to validator init if a new validator is created.

name = u'validator'

schema()

validate(*value*)
This is just to unit test backwards compatibility of json schema with old validators

flexget.validator.**any_schema**(*schemas*)
Creates a schema that will match any of the given schemas. Will not use anyOf if there is just one validator in the list, for simpler error messages.

flexget.validator.**build_options_validator**(*options*)

flexget.validator.**complex_test**()

flexget.validator.**factory**(*name=u'root'*, ***kwargs*)
Factory method, returns validator instance.

2.1.16 Subpackages

utils Package

bittorrent Module

Torrenting utils, mostly for handling bencoding and torrent files.

class flexget.utils.bittorrent.**Torrent**(*content*)
Bases: object

Represents a torrent

KEY_TYPE
alias of str

add_multitracker(*tracker*)
Appends multi-tracker to this torrent

comment

encode()

classmethod from_file(*filename*)
Create torrent from file on disk.

get_filelist()
Return array containing fileinfo dictionaries (name, length, path)

info_hash
Return Torrent info hash

private

remove_multitracker(*tracker*)
Removes passed multi-tracker from this torrent

size
Return total size of the torrent

trackers

Returns List of trackers, supports single-tracker and multi-tracker implementations

flexget.utils.bittorrent.**bdecode**(*text*)

flexget.utils.bittorrent.**bencode**(*data*)

flexget.utils.bittorrent.**clean_meta**(*meta*, *including_info=False*, *logger=None*)
Clean meta dict. Optionally log changes using the given logger.

See also http://packages.python.org/pyrocore/apidocs/pyrocore.util.metafile-pysrc.html#clean_meta

@param logger: If given, a callable accepting a string message. @return: Set of keys removed from C{meta}.

flexget.utils.bittorrent.**decode_item**(*next*, *token*)

flexget.utils.bittorrent.**encode_dictionary**(*data*)

flexget.utils.bittorrent.**encode_integer**(*data*)

flexget.utils.bittorrent.**encode_list**(*data*)

flexget.utils.bittorrent.**encode_string**(*data*)

flexget.utils.bittorrent.**encode_unicode**(*data*)

flexget.utils.bittorrent.**is_torrent_file**(*metafilepath*)
Check whether a file looks like a metafile by peeking into its content.

Note that this doesn't ensure that the file is a complete and valid torrent, it just allows fast filtering of candidate files.

@param metafilepath: Path to the file to check, must have read permissions for it. @return: True if there is a high probability this is a metafile.

flexget.utils.bittorrent.**tokenize**(*text*, *match=<built-in method match of _sre.SRE_Pattern object>*)

cached_input Module

class flexget.utils.cached_input.**InputCache**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

added

entries

hash

id

name

class flexget.utils.cached_input.**InputCacheEntry**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

cache_id

entry

id

class flexget.utils.cached_input.**cached**(*name*, *persist=None*)
Bases: object

Implements transparent caching decorator @cached for inputs.

Decorator has two parameters:

- **name** in which the configuration is present in tasks configuration.
- **key** in which the configuration has the cached resource identifier (ie. url). If the key is not given or present in the configuration :name: is expected to be a cache name (ie. url)

Note: Configuration assumptions may make this unusable in some (future) inputs

cache = TimedDict({})

flexget.utils.cached_input.**config_hash**(*config*)

Parameters **config** (*dict*) – Configuration

Returns MD5 hash for *config*

flexget.utils.cached_input.**db_cleanup**(*manager*, *session*)
Removes old input caches from plugins that are no longer configured.

database Module

class flexget.utils.database.**CaseInsensitiveWord**(*word*)
Bases: sqlalchemy.ext.hybrid.Comparator

Hybrid value representing a string that compares case insensitively.

lower()

operate(*op*, *other*)

flexget.utils.database.**ignore_case_property**(*text_attr*)

flexget.utils.database.**pipe_list_synonym**(*name*)
Converts pipe separated text into a list

flexget.utils.database.**quality_property**(*text_attr*)

flexget.utils.database.**quality_requirement_property**(*text_attr*)

flexget.utils.database.**safe_pickle_synonym**(*name*)
Used to store Entry instances into a PickleType column in the database.

In order to ensure everything can be loaded after code changes, makes sure no custom python classes are pickled.

flexget.utils.database.**text_date_synonym**(*name*)
Converts Y-M-D date strings into datetime objects

flexget.utils.database.**with_session**(**args*, ***kwargs*)
” A decorator which creates a new session if one was not passed via keyword argument to the function.

Automatically commits and closes the session if one was created, caller is responsible for commit if passed in.

If arguments are given when used as a decorator, they will automatically be passed to the created Session when one is not supplied.

flexget.utils.database.**year_property**(*date_attr*)

imdb Module

class flexget.utils.imdb.**ImdbParser**
Bases: object

Quick-hack to parse relevant imdb details

parse(*imdb_id*)

class flexget.utils.imdb.**ImdbSearch**
Bases: object

best_match(*name*, *year=None*)
Return single movie that best matches name criteria or None

ireplace(*text*, *old*, *new*, *count=0*)
Case insensitive string replace

search(*name*)
Return array of movie details (dict)

smart_match(*raw_name*)
Accepts messy name, cleans it and uses information available to make smartest and best match

flexget.utils.imdb.**extract_id**(*url*)
Return IMDb ID of the given URL. Return None if not valid or if URL is not a string.

flexget.utils.imdb.**is_imdb_url**(*url*)
Tests the url to see if it's for imdb.com.

flexget.utils.imdb.**make_url**(*imdb_id*)
Return IMDb URL of the given ID

log Module

Logging utilities

class flexget.utils.log.**LogMessage**(*md5sum*)
Bases: sqlalchemy.ext.declarative.api.Base

Declarative

added

id

md5sum

flexget.utils.log.**purge**(*manager*, *session*)
Purge old messages from database

qualities Module

class flexget.utils.qualities.**Quality**(*text=u''*)
Bases: object

Parses and stores the quality of an entry in the four component categories.

components

name

parse(*text*)
Parses a string to determine the quality in the four component categories.

Parameters **text** – The string to parse

class flexget.utils.qualities.**QualityComponent**(*type*, *value*, *name*, *regexp=None*, *modifier=None*, *defaults=None*)
Bases: object

matches(*text*)
Test if quality matches to text.

Parameters **text** (*string*) – data te be tested against

Returns tuple (matches, remaining text without quality data)

class flexget.utils.qualities.**RequirementComponent**(*type*)
Bases: object

Represents requirements for a given component type. Can evaluate whether a given QualityComponent meets those requirements.

add_requirement(*text*)

allows(*comp*, *loose=False*)

reset()

class flexget.utils.qualities.**Requirements**(*req=u''*)
Bases: object

Represents requirements for allowable qualities. Can determine whether a given Quality passes requirements.

allows(*qual*, *loose=False*)
Determine whether this set of requirements allows a given quality.

Parameters

- **qual** (Quality) – The quality to evaluate.
- **loose** (*bool*) – If True, only ! (not) requirements will be enforced.

Return type bool

Returns True if given quality passes all component requirements.

components

parse_requirements(*text*)
Parses a requirements string.

Parameters **text** – The string containing quality requirements.

flexget.utils.qualities.**all_components**()

flexget.utils.qualities.**get**(*quality_name*)
Returns a quality object based on canonical quality name.

requests Module

class flexget.utils.requests.**Session**(*timeout=30*, *max_retries=1*)
Bases: requests.sessions.Session

Subclass of requests Session class which defines some of our own defaults, records unresponsive sites, and raises errors by default.

add_cookiejar(*cookiejar*)
Merges cookies from *cookiejar* into cookiejar for this session.

Parameters **cookiejar** – CookieJar instance to add to the session.

request(*method*, *url*, **args*, ***kwargs*)
Does a request, but raises Timeout immediately if site is known to timeout, and records sites that timeout. Also raises errors getting the content by default.

Parameters **raise_status** (*bool*) – If True, non-success status code responses will be raised as errors (True by default)

set_domain_delay(*domain*, *delay*)
Registers a minimum interval between requests to *domain*

Parameters

- **domain** – The domain to set the interval on
- **delay** – The amount of time between requests, can be a timedelta or string like '3 seconds'

flexget.utils.requests.**get**(*url*, ***kwargs*)
Sends a GET request. Returns `Response` object.

Parameters

- **url** – URL for the new `Request` object.
- **kwargs** – Optional arguments that `request` takes.

flexget.utils.requests.**is_unresponsive**(*url*)
Checks if host of given url has timed out within WAIT_TIME

Parameters **url** – The url to check

Returns True if the host has timed out within WAIT_TIME

Return type bool

flexget.utils.requests.**post**(*url*, *data=None*, ***kwargs*)
Sends a POST request. Returns `Response` object.

Parameters

- **url** – URL for the new `Request` object.
- **data** – (optional) Dictionary or bytes to send in the body of the `Request`.
- **kwargs** – Optional arguments that `request` takes.

flexget.utils.requests.**request**(*method*, *url*, ***kwargs*)

flexget.utils.requests.**set_unresponsive**(*url*)
Marks the host of a given url as unresponsive

Parameters **url** – The url that timed out

flexget.utils.requests.**wait_for_domain**(*url*, *delay_dict*)

search Module

Common tools used by plugins implementing search plugin api

flexget.utils.search.**clean_symbols**(*text*)
Replaces common symbols with spaces. Also normalize unicode strings in decomposed form.

flexget.utils.search.**clean_title**(*title*)
Removes common codec, sound keywords, and special characters info from titles to facilitate loose title comparison.

flexget.utils.search.**normalize_unicode**(*text*)

flexget.utils.search.**torrent_availability**(*seeds*, *leeches*)
Returns a rating based on seeds and leeches for a given torrent.

Parameters

- **seeds** – Number of seeds on the torrent
- **leeches** – Number of leeches on the torrent

Returns A numeric rating

simple_persistence Module

NOTE:

Avoid using this module on your own or in plugins, this was originally made for 0.9 -> 1.0 transition.

You can safely use task.simple_persistence and manager.persist, if we implement something better we can replace underlying mechanism in single point (and provide transparent switch).

class flexget.utils.simple_persistence.**SimpleKeyValue**(*task*, *plugin*, *key*, *value*)
Bases: sqlalchemy.ext.declarative.api.Base

added

id

key

plugin

task

value

class flexget.utils.simple_persistence.**SimplePersistence**(*plugin=None*)
Bases: _abcoll.MutableMapping

Store simple values that need to be persisted between FlexGet runs. Interface is like a *dict*.

This should only be used if a plugin needs to store a few values, otherwise it should create a full table in the database.

class_store = defaultdict(<function <lambda> at 0x7f56c285a5f0>, {})

classmethod flush(*task=None*)
Flush all in memory key/values to database.

classmethod load(*task=None*)
Load all key/values from *task* into memory from database.

store

class flexget.utils.simple_persistence.**SimpleTaskPersistence**(*task*)
Bases: *flexget.utils.simple_persistence.SimplePersistence*

plugin

flexget.utils.simple_persistence.**db_cleanup**(*manager*, *session*)
Clean up values in the db from tasks which no longer exist.

flexget.utils.simple_persistence.**flush_task**(*task*)
Stores all in memory key/value pairs to database when a task has completed.

flexget.utils.simple_persistence.**flush_taskless**(*manager*)

flexget.utils.simple_persistence.**load_task**(*task*)
Loads all key/value pairs into memory before a task starts.

flexget.utils.simple_persistence.**load_taskless**(*manager*)
Loads all key/value pairs into memory which aren't associated with a specific task.

soup Module

flexget.utils.soup.**get_soup**(*obj*, *parser=u'html5lib'*)

sqlalchemy_utils Module

Miscellaneous SQLAlchemy helpers.

class flexget.utils.sqlalchemy_utils.**ContextSession**(*bind=None*, *autoflush=True*, *expire_on_commit=True*, *_enable_transaction_accounting=True*, *autocommit=False*, *twophase=False*, *weak_identity_map=True*, *binds=None*, *extension=None*, *info=None*, *query_cls=<class 'sqlalchemy.orm.query.Query'>*)
Bases: sqlalchemy.orm.session.Session

sqlalchemy.orm.Session which can be used as context manager

flexget.utils.sqlalchemy_utils.**create_index**(*table_name*, *session*, **column_names*)
Creates an index on specified *columns* in *table_name*

Parameters

- **table_name** – Name of table to create the index on.
- **session** – Session object which should be used
- **column_names** – The names of the columns that should belong to this index.

flexget.utils.sqlalchemy_utils.**drop_tables**(*names*, *session*)
Takes a list of table names and drops them from the database if they exist.

flexget.utils.sqlalchemy_utils.**get_index_by_name**(*table*, *name*)
Find declaratively defined index from table by name

Parameters

- **table** – Table object
- **name** (*string*) – Name of the index to get

Returns Index object

flexget.utils.sqlalchemy_utils.**table_add_column**(*table*, *name*, *col_type*, *session*, *default=None*)
Adds a column to a table

> **Warning:** Uses raw statements, probably needs to be changed in order to work on other databases besides SQLite

Parameters

- **table** (*string*) – Table to add column to (can be name or schema)
- **name** (*string*) – Name of new column to add
- **col_type** – The sqlalchemy column type to add
- **session** (Session) – SQLAlchemy Session to do the alteration
- **default** – Default value for the created column (optional)

flexget.utils.sqlalchemy_utils.**table_columns**(*table*, *session*)

Parameters

- **table** (*string*) – Name of table or table schema
- **session** (Session) – SQLAlchemy Session

Returns List of column names in the table or empty list

flexget.utils.sqlalchemy_utils.**table_exists**(*name*, *session*)

Use SQLAlchemy reflect to check table existences.

Parameters

- **name** (*string*) – Table name to check
- **session** (Session) – Session to use

Returns True if table exists, False otherwise

Return type bool

flexget.utils.sqlalchemy_utils.**table_schema**(*name*, *session*)

Returns Table schema using SQLAlchemy reflect as it currently exists in the db

Return type Table

template Module

class flexget.utils.template.**FlexGetTemplate**

Bases: jinja2.environment.Template

Adds lazy lookup support when rendering templates.

new_context(*vars=None*, *shared=False*, *locals=None*)

exception flexget.utils.template.**RenderError**

Bases: exceptions.Exception

Error raised when there is a problem with jinja rendering.

flexget.utils.template.**filter_d**(*value*, *default_value=u''*, *boolean=False*)

flexget.utils.template.**filter_date_suffix**(*date*)

flexget.utils.template.**filter_default**(*value*, *default_value=u''*, *boolean=False*)

flexget.utils.template.**filter_format_number**(*val*, *places=None*, *grouping=True*)

Formats a number according to the user's locale.

flexget.utils.template.**filter_formatdate**(*val*, *format*)

Returns a string representation of a datetime object according to format string.

flexget.utils.template.**filter_pad**(*val*, *width*, *fillchar=u'0'*)

Pads a number or string with fillchar to the specified width.

flexget.utils.template.**filter_parsedate**(*val*)
Attempts to parse a date according to the rules in RFC 2822

flexget.utils.template.**filter_pathbase**(*val*)
Base name of a path.

flexget.utils.template.**filter_pathdir**(*val*)
Directory containing the given path.

flexget.utils.template.**filter_pathext**(*val*)
Extension of a path (including the '.').

flexget.utils.template.**filter_pathname**(*val*)
Base name of a path, without its extension.

flexget.utils.template.**filter_pathscrub**(*val*, *os_mode=None*)
Replace problematic characters in a path.

flexget.utils.template.**filter_re_replace**(*val*, *pattern*, *repl*)
Perform a regexp replacement on the given string.

flexget.utils.template.**filter_re_search**(*val*, *pattern*)
Perform a search for given regexp pattern, return the matching portion of the text.

flexget.utils.template.**filter_to_date**(*date_time_val*)

flexget.utils.template.**get_template**(*templatename*, *pluginname=None*)
Loads a template from disk. Looks in both included plugins and users custom plugin dir.

flexget.utils.template.**make_environment**(*manager*)
Create our environment and add our custom filters

flexget.utils.template.**now**()

flexget.utils.template.**render**(*template*, *context*)
Renders a Template with *context* as its context.

Parameters

- **template** – Template or template string to render.
- **context** – Context to render the template from.

Returns The rendered template text.

flexget.utils.template.**render_from_entry**(*template_string*, *entry*)
Renders a Template or template string with an Entry as its context.

flexget.utils.template.**render_from_task**(*template*, *task*)
Renders a Template with a task as its context.

Parameters

- **template** – Template or template string to render.
- **task** – Task to render the template from.

Returns The rendered template text.

tools Module

Contains miscellaneous helpers

class flexget.utils.tools.**BufferQueue**(*maxsize=0*)
Bases: Queue.Queue

Used in place of a file-like object to capture text and access it safely from another thread.

exception Empty
Bases: exceptions.Exception

Exception raised by Queue.get(block=0)/get_nowait().

BufferQueue.**write**(*line*)

exception flexget.utils.tools.**MergeException**(*value*)
Bases: exceptions.Exception

class flexget.utils.tools.**ReList**(**args*, ***kwargs*)
Bases: list

A list that stores regexps.

You can add compiled or uncompiled regexps to the list. It will always return the compiled version. It will compile the text regexps on demand when first accessed.

flags = 34

class flexget.utils.tools.**SmartRedirectHandler**
Bases: urllib2.HTTPRedirectHandler

http_error_301(*req*, *fp*, *code*, *msg*, *headers*)

http_error_302(*req*, *fp*, *code*, *msg*, *headers*)

class flexget.utils.tools.**TimedDict**(*cache_time=u'5 minutes'*)
Bases: _abcoll.MutableMapping

Acts like a normal dict, but keys will only remain in the dictionary for a specified time span.

flexget.utils.tools.**arithmeticEval**(*s*)
A safe eval supporting basic arithmetic operations.

Parameters **s** – expression to evaluate

Returns value

flexget.utils.tools.**console**(*text*)
Print to console safely.

flexget.utils.tools.**convert_bytes**(*bytes*)
Returns given bytes as prettified string.

flexget.utils.tools.**decode_html**(*value*)

Parameters **value** (*string*) – String to be html-decoded

Returns Html decoded string

flexget.utils.tools.**encode_html**(*unicode_data*, *encoding=u'ascii'*)
Encode unicode_data for use as XML or HTML, with characters outside of the encoding converted to XML numeric character references.

flexget.utils.tools.**merge_dict_from_to**(*d1*, *d2*)
Merges dictionary d1 into dictionary d2. d1 will remain in original form.

flexget.utils.tools.**multiply_timedelta**(*interval*, *number*)
timedeltas can not normally be multiplied by floating points. This does that.

flexget.utils.tools.**parse_timedelta**(*value*)
Parse a string like '5 days' into a timedelta object. Also allows timedeltas to pass through.

flexget.utils.tools.**pid_exists**(*pid*)
Check whether pid exists in the current process table.

flexget.utils.tools.**singleton**(*cls*)

flexget.utils.tools.**str_to_boolean**(*string*)

flexget.utils.tools.**str_to_int**(*string*)

flexget.utils.tools.**strip_html**(*text*)
Tries to strip all HTML tags from *text*. If unsuccessful returns original text.

flexget.utils.tools.**urlopener**(*url_or_request*, *log*, ***kwargs*)
Utility function for pulling back a url, with a retry of 3 times, increasing the timeout, etc. Re-raises any errors as URLError.

> **Warning:** This is being replaced by requests library. flexget.utils.requests should be used going forward.

Parameters

- **url_or_request** (*str*) – URL or Request object to get.
- **log** – Logger to log debug info and errors to
- **kwargs** – Keyword arguments to be passed to urlopen

Returns The file-like object returned by urlopen

Subpackages

titles Package

titles Package

movie Module

class flexget.utils.titles.movie.**MovieParser**
Bases: *flexget.utils.titles.parser.TitleParser*

is_movie

is_series

parse(*data=None*)
Parse movie name. Populates name, year, quality and proper_count attributes

proper

reset()

valid

flexget.utils.titles.movie.**diff_pos**(*string1*, *string2*)
Returns first position where string1 and string2 differ.

parser Module

class flexget.utils.titles.parser.**TitleParser**

Bases: object

codecs = [u'x264', u'x.264', u'h264', u'h.264', u'XViD']

cutoffs = [u'limited', u'xvid', u'h264', u'x264', u'h.264', u'x.264', u'screener', u'unrated', u'3d', u'extended', u'directo

editions = [u'dc', u'extended', u'uncut', u'remastered', u'unrated', u'theatrical', u'chrono', u'se']

static **ireplace**(*data*, *old*, *new*, *count=0*, *not_in_word=False*)

Case insensitive string replace

propers = [u'proper', u'repack', u'rerip', u'real', u'final']

static **re_not_in_word**(*regexp*)

remove = [u'imax']

static **remove_words**(*text*, *words*, *not_in_word=False*)

Clean all given :words: from :text: case insensitively

sounds = [u'AC3', u'DD5.1', u'DTS']

specials = [u'special', u'bonus', u'extra', u'omake', u'ova']

static **strip_spaces**(*text*)

Removes all unnecessary duplicate spaces from a text

series Module

class flexget.utils.titles.series.**SeriesParser**(*name=None*, *alternate_names=None*, *identified_by=u'auto'*, *name_regexps=None*, *ep_regexps=None*, *date_regexps=None*, *sequence_regexps=None*, *id_regexps=None*, *strict_name=False*, *allow_groups=None*, *allow_seasonless=True*, *date_dayfirst=None*, *date_yearfirst=None*, *special_ids=None*, *prefer_specials=False*, *assume_special=False*)

Bases: *flexget.utils.titles.parser.TitleParser*

Parse series.

Name series name

Data data to parse

Expect_ep expect series to be in season, ep format (ep_regexps)

Expect_id expect series to be in id format (id_regexps)

clean_regexps = [u'\\[.*?\\]', u'\\(.*?\\)']

date_regexps = [u'(?<![^\\W_])(\\d{2,4})[/ -](\\d{1,2})[/ -](\\d{1,2})(?![^\\W_])', u'(?<![^\\W_])(\\d{1,2})[/ -](\\d{1,2})[/ -

english_numbers = [u'one', u'two', u'three', u'four', u'five', u'six', u'seven', u'eight', u'nine', u'ten']

ep_regexps = [u'(?<![^\\W_])(?:series|season|s)\\s?(\\d{1,4})(?:\\s(?:.*\\s)?)?(?:episode|ep|e|part|pt)\\s?(\\d{1,3}|X{0,3}(?:

guess_name()

This will attempt to guess a series name based on the provided data.

id_regexps = []

identifier
Return String identifier for parsed episode, eg. S01E02 (will be the first identifier if this is a pack)

identifiers
Return all identifiers this parser represents. (for packs)

ignore_prefixes = [u'(?:\\[[^\\[\\]]*\\])', u'(?:HD.720p?:)', u'(?:HD.1080p?:)']

is_movie

is_series

pack_identifier
Return a combined identifier for the whole pack if this has more than one episode.

parse(*data=None*, *field=None*, *quality=None*)

parse_date(*data*)
Parses :data: for a date identifier. If found, returns the date and regexp match object If no date is found returns False

parse_episode(*data*)
Parses :data: for an episode identifier. If found, returns a dict with keys for season, episode, end_episode and the regexp match object If no episode id is found returns False

parse_unwanted(*data*)
Parses data for an unwanted hits. Return True if the data contains unwanted hits.

parse_unwanted_sequence(*data*)
Parses data for an unwanted id hits. Return True if the data contains unwanted hits.

proper

regexp = u'(?:pt|part)\\s?(\\d+|X{0,3}(?:IX|XI{0,4}|VI{0,4}|IV|V|I{1,4}))'

remove_dirt(*data*)
Replaces some characters with spaces

roman_numeral_re = u'X{0,3}(?:IX|XI{0,4}|VI{0,4}|IV|V|I{1,4})'

roman_to_int(*roman*)
Converts roman numerals up to 39 to integers

separators = u'[/ -]'

sequence_regexps = [u'(?<![^\\W_])(\\d{1,3})(?:v(?P<version>\\d))?(?![^\\W_])', u'(?<![^\\W_])(?:pt|part)\\s?(\\d+|X{

unwanted_regexps = [u'(\\d{1,3})\\s?x\\s?(0+)[^1-9]', u'S(\\d{1,3})D(\\d{1,3})', u'(\\d{1,3})\\s?x\\s?(all)', u'(?:season(?:s

unwanted_sequence_regexps = [u'seasons?\\s?\\d{1,2}']

CHAPTER 3

Plugins

This list is not up-to-date and should probably only contain plugins that serve API to other plugins.

3.1 plugins Package

3.1.1 plugins Package

Standard plugin package.

3.1.2 api_tmdb Module

class flexget.plugins.api_tmdb.**ApiTmdb**
Bases: object

Does lookups to TMDb and provides movie information. Caches lookups.

static get_movie_details(*movie*, *session*, *result=None*)
Populate details for this :movie: from TMDb

static lookup(**args*, ***kwargs*)

class flexget.plugins.api_tmdb.**TMDBContainer**(*init_object=None*)
Bases: object

Base class for TMDb objects

update_from_dict(*update_dict*)
Populates any simple (string or number) attributes from a dict

update_from_object(*update_object*)
Populates any simple (string or number) attributes from object attributes

class flexget.plugins.api_tmdb.**TMDBGenre**(*init_object=None*)
Bases: *flexget.plugins.api_tmdb.TMDBContainer*, sqlalchemy.ext.declarative.api.Base

id

name

class flexget.plugins.api_tmdb.**TMDBMovie**(*init_object=None*)
Bases: *flexget.plugins.api_tmdb.TMDBContainer*, sqlalchemy.ext.declarative.api.Base

adult

alternative_name

budget

certification

genres

homepage

id

imdb_id

language

movie_type

name

original_name

overview

popularity

posters

rating

released

revenue

runtime

tagline

trailer

translated

update_from_object(*update_object*)

updated

url

votes

year = <sqlalchemy.sql.elements.Extract object>

class flexget.plugins.api_tmdb.**TMDBPoster**(*init_object=None*)

Bases: *flexget.plugins.api_tmdb.TMDBContainer*, sqlalchemy.ext.declarative.api.Base

db_id

file

get_file(*only_cached=False*)

Makes sure the poster is downloaded to the local cache (in userstatic folder) and returns the path split into a list of directory and file components

movie_id

size

url

class flexget.plugins.api_tmdb.**TMDBSearchResult**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

id

movie

movie_id

search

flexget.plugins.api_tmdb.**register_plugin**()

3.1.3 api_tvdb Module

3.1.4 plugin_change_warn Module

class flexget.plugins.plugin_change_warn.**ChangeWarn**
Bases: object

Gives warning if user has deprecated / changed configuration in the root level.

Will be replaced by root level validation in the future!

Contains ugly hacks, better to include all deprecation warnings here during 1.0 BETA phase

on_task_start(*task*, *config*)

flexget.plugins.plugin_change_warn.**register_plugin**()

3.1.5 plugin_cookies Module

class flexget.plugins.plugin_cookies.**PluginCookies**
Adds cookie to all requests (rss, resolvers, download). Anything that uses urllib2 to be exact.

Currently supports Firefox 3 cookies only.

Example:

```
cookies: /path/firefox/profile/something/cookies.sqlite
```

cookiejars = TimedDict({})

on_task_abort(*task*, *config*)
Task exiting, remove cookiejar

on_task_exit(*task*, *config*)
Task exiting, remove cookiejar

on_task_start(*task*, *config*)
Task starting, install cookiejar

prepare_config(*config*)

schema = {u'oneOf': [{u'type': u'string', u'format': u'file'}, {u'additionalProperties': False, u'type': u'object', u'proper

sqlite2cookie(*filename*)

flexget.plugins.plugin_cookies.**register_plugin**()

3.1.6 plugin_deluge Module

class flexget.plugins.plugin_deluge.**DelugePlugin**
Bases: object

Base class for deluge plugins, contains settings and methods for connecting to a deluge daemon.

on_task_abort(*task*, *config*)

on_task_start(*task*, *config*)
Raise a DependencyError if our dependencies aren't available

prepare_connection_info(*config*)

class flexget.plugins.plugin_deluge.**InputDeluge**
Bases: *flexget.plugins.plugin_deluge.DelugePlugin*

Create entries for torrents in the deluge session.

on_connect_success(*result*, *task*, *config*)
Creates a list of FlexGet entries from items loaded in deluge and stores them to self.entries

on_task_input(*task*, *config*)
Generates and returns a list of entries from the deluge daemon.

prepare_config(*config*)

schema = {u'anyOf': [{u'type': u'boolean'}, {u'additionalProperties': False, u'type': u'object', u'properties': {u'userna

settings_map = {u'files': (u'content_files', <function <lambda> at 0x7f56bf5b1c08>), u'hash': u'torrent_info_hash', u

class flexget.plugins.plugin_deluge.**OutputDeluge**
Bases: *flexget.plugins.plugin_deluge.DelugePlugin*

Add the torrents directly to deluge, supporting custom save paths.

add_to_deluge11(*task*, *config*)
Add torrents to deluge using deluge 1.1.x api.

on_connect_success(*result*, *task*, *config*)
Gets called when successfully connected to a daemon.

on_task_abort(*task*, *config*)
Make sure normal cleanup tasks still happen on abort.

on_task_download(*task*, *config*)
Call download plugin to generate the temp files we will load into deluge then verify they are valid torrents

on_task_exit(*task*, *config*)
Make sure all temp files are cleaned up when task exits

on_task_output(*task*, *config*)
Add torrents to deluge at exit.

on_task_start(*task*, *config*)
Detect what version of deluge is loaded.

prepare_config(*config*)

schema = {u'anyOf': [{u'type': u'boolean'}, {u'additionalProperties': False, u'type': u'object', u'properties': {u'automa

flexget.plugins.plugin_deluge.**add_deluge_windows_install_dir_to_sys_path**()

flexget.plugins.plugin_deluge.**register_plugin**()

3.1.7 plugin_entry_trace Module

class flexget.plugins.plugin_entry_trace.**EntryOperations**
Bases: object

Records accept, reject and fail metainfo into entries.

Creates fields:

```
accepted_by: <plugin name>
rejected_by: <plugin name>
failed_by: <plugin name>

reason: <given message by plugin>
```

on_task_input(*task*, *config*)

flexget.plugins.plugin_entry_trace.**on_entry_action**(*entry*, *act=None*, *task=None*, *reason=None*, ***kwargs*)

flexget.plugins.plugin_entry_trace.**register_plugin**()

3.1.8 plugin_formlogin Module

class flexget.plugins.plugin_formlogin.**FormLogin**
Bases: object

Login on form

on_task_abort(*task*, *config*)
Task exiting, remove cookiejar

on_task_exit(*task*, *config*)
Task exiting, remove cookiejar

on_task_start(*task*, *config*)

schema = {u'additionalProperties': False, u'required': [u'url', u'username', u'password'], u'type': u'object', u'properti

flexget.plugins.plugin_formlogin.**register_plugin**()

3.1.9 plugin_import_series Module

3.1.10 plugin_include Module

class flexget.plugins.plugin_include.**PluginInclude**
Bases: object

Include configuration from another yaml file.

Example:

```
include: series.yml
```

File content must be valid for a task configuration

on_task_start(*task*, *config*)

schema = {u'oneOf': [{u'minItems': 1, u'items': {u'type': u'string', u'title': u'single value'}, u'type': u'array', u'title': u

flexget.plugins.plugin_include.**register_plugin**()

3.1.11 plugin_preset Module

3.1.12 plugin_sort_by Module

class flexget.plugins.plugin_sort_by.**PluginSortBy**
Bases: object

Sort task entries based on a field

Example:

```
sort_by: title
```

More complex:

```
sort_by:
  field: imdb_score
  reverse: yes
```

Reverse the order of the entries, without sorting on a field:

```
sort_by:
  reverse: yes
```

on_task_filter(*task*, *config*)

schema = {u'oneOf': [{u'type': u'string'}, {u'additionalProperties': False, u'type': u'object', u'properties': {u'field': {u

flexget.plugins.plugin_sort_by.**register_plugin**()

3.1.13 plugin_spy_headers Module

class flexget.plugins.plugin_spy_headers.**CustomHTTPConnection**(**args*, ***kwargs*)
Bases: httplib.HTTPConnection

putheader(*header*, *value*)

class flexget.plugins.plugin_spy_headers.**HTTPCaptureHeaderHandler**(*debuglevel=0*)
Bases: urllib2.AbstractHTTPHandler

do_open(*http_class*, *req*)

handler_order = 400

http_open(*req*)

http_request(*request*)

https_open(*req*)

https_request(*request*)

class flexget.plugins.plugin_spy_headers.**PluginSpyHeaders**
Bases: object

Logs all headers sent in http requests. Useful for resolving issues.

WARNING: At the moment this modifies requests somehow!

static log_requests_headers(*response*, ***kwargs*)

on_task_abort(*task*, *config*)
Task exiting, remove additions

on_task_exit(*task*, *config*)
Task exiting, remove additions

on_task_start(*task*, *config*)

schema = {u'type': u'boolean'}

flexget.plugins.plugin_spy_headers.**register_plugin**()

3.1.14 plugin_transmission Module

class flexget.plugins.plugin_transmission.**PluginTransmission**
Bases: *flexget.plugins.plugin_transmission.TransmissionBase*

Add url from entry url to transmission

Example:

```
transmission:
  host: localhost
  port: 9091
  netrc: /home/flexget/.tmnetrc
  username: myusername
  password: mypassword
  path: the download location
```

Default values for the config elements:

```
transmission:
  host: localhost
  port: 9091
  enabled: yes
```

add_to_transmission(*cli*, *task*, *config*)
Adds accepted entries to transmission

on_task_abort(*task*, *config*)
Make sure all temp files are cleaned up when task exits

on_task_download(*task*, *config*)
Call download plugin to generate the temp files we will load into deluge then verify they are valid torrents

on_task_exit(*task*, *config*)
Make sure all temp files are cleaned up when task exits

on_task_output(**args*, ***kwargs*)

prepare_config(*config*)

schema = {u'anyOf': [{u'type': u'boolean'}, {u'additionalProperties': False, u'type': u'object', u'properties': {u'bandw

class flexget.plugins.plugin_transmission.**PluginTransmissionClean**
Bases: *flexget.plugins.plugin_transmission.TransmissionBase*

Remove completed torrents from Transmission.

Examples:

```
clean_transmission: yes  # ignore both time and ratio

clean_transmission:      # uses transmission's internal limits for idle time and seed ratio ( if
  transmission_seed_limits: yes
```

```
clean_transmission:      # matches time only
  finished_for: 2 hours

clean_transmission:      # matches ratio only
  min_ratio: 0.5

clean_transmission:      # matches time OR ratio
  finished_for: 2 hours
  min_ratio: 0.5
```

Default values for the config elements:

```
clean_transmission:
  host: localhost
  port: 9091
  enabled: yes
```

on_task_exit(*task*, *config*)

validator()
Return config validator

class flexget.plugins.plugin_transmission.**PluginTransmissionInput**
Bases: *flexget.plugins.plugin_transmission.TransmissionBase*

on_task_input(*task*, *config*)

prepare_config(*config*)

validator()
Return config validator

class flexget.plugins.plugin_transmission.**TransmissionBase**
Bases: object

check_seed_limits(*torrent*, *session*)

create_rpc_client(*config*)

on_task_start(**args*, ***kwargs*)

prepare_config(*config*)

torrent_info(*torrent*, *config*)

flexget.plugins.plugin_transmission.**register_plugin**()

flexget.plugins.plugin_transmission.**save_opener**(*f*)
Transmissionrpc sets a new default opener for urllib2 We use this as a decorator to capture and restore it when needed

3.1.15 plugin_try_regexp Module

class flexget.plugins.plugin_try_regexp.**PluginTryRegexp**
Bases: object

This plugin allows user to test regexps for a task.

matches(*entry*, *regexp*)
Return True if any of the entry string fields match given regexp

on_task_filter(*task*, *config*)

flexget.plugins.plugin_try_regexp.**register_parser_arguments**()

flexget.plugins.plugin_try_regexp.**register_plugin**()

3.1.16 plugin_urlrewriting Module

class flexget.plugins.plugin_urlrewriting.**DisableUrlRewriter**
Bases: object

Disable certain urlrewriters.

on_task_abort(*task*, *config*)

on_task_exit(*task*, *config*)

on_task_start(*task*, *config*)

schema = {u'items': {u'type': u'string'}, u'type': u'array'}

class flexget.plugins.plugin_urlrewriting.**PluginUrlRewriting**
Bases: object

Provides URL rewriting framework

on_task_urlrewrite(*task*, *config*)

url_rewritable(*task*, *entry*)
Return True if entry is urlrewritable by registered rewriter.

url_rewrite(*task*, *entry*)
Rewrites given entry url. Raises UrlRewritingError if failed.

exception flexget.plugins.plugin_urlrewriting.**UrlRewritingError**(*value*)
Bases: exceptions.Exception

flexget.plugins.plugin_urlrewriting.**register_plugin**()

3.1.17 plugin_verbose Module

class flexget.plugins.plugin_verbose.**Verbose**
Bases: object

Verbose entry accept, reject and failure

on_task_exit(*task*, *config*)

on_task_metainfo(*task*, *config*)

verbose_details(*entry*, *task=None*, *act=None*, *reason=None*, ***kwargs*)

flexget.plugins.plugin_verbose.**register_parser_arguments**()

flexget.plugins.plugin_verbose.**register_plugin**()

3.1.18 plugin_verbose_details Module

class flexget.plugins.plugin_verbose_details.**NoEntriesOk**
Bases: object

Allows manually silencing the warning message for tasks that regularly produce no entries.

on_task_start(*task*, *config*)

schema = {u'type': u'boolean'}

class flexget.plugins.plugin_verbose_details.**PluginDetails**
Bases: object

on_task_download(*task*, *config*)

on_task_input(*task*, *config*)

on_task_start(*task*, *config*)

flexget.plugins.plugin_verbose_details.**register_plugin**()

3.1.19 search_kat Module

class flexget.plugins.search_kat.**SearchKAT**
Bases: object

KAT search plugin.

should accept: kat:

category: <category> verified: yes/no

categories: all movies tv music books xxx other

schema = {u'additionalProperties': False, u'type': u'object', u'properties': {u'category': {u'enum': [u'all', u'movies', u'

search(*task*, *entry*, *config*)

flexget.plugins.search_kat.**register_plugin**()

3.1.20 search_rss Module

class flexget.plugins.search_rss.**SearchRSS**
Bases: object

A generic search plugin that can use rss based search feeds. Configure it like rss plugin, but include {{{search_term}}} in the url where the search term should go.

schema = {u'$ref': u'/schema/plugin/rss'}

search(*task*, *entry*, *config=None*)

flexget.plugins.search_rss.**register_plugin**()

3.1.21 urlrewrite_bakabt Module

class flexget.plugins.urlrewrite_bakabt.**UrlRewriteBakaBT**
Bases: object

BakaBT urlrewriter.

parse_download_page(**args*, ***kwargs*)

url_rewritable(*task*, *entry*)

url_rewrite(*task*, *entry*)

flexget.plugins.urlrewrite_bakabt.**register_plugin**()

3.1.22 urlrewrite_btchat Module

class flexget.plugins.urlrewrite_btchat.**UrlRewriteBtChat**
Bases: object

BtChat urlrewriter.

url_rewritable(*task*, *entry*)

url_rewrite(*task*, *entry*)

flexget.plugins.urlrewrite_btchat.**register_plugin**()

3.1.23 urlrewrite_btjunkie Module

class flexget.plugins.urlrewrite_btjunkie.**UrlRewriteBtJunkie**
Bases: object

BtJunkie urlrewriter.

url_rewritable(*task*, *entry*)

url_rewrite(*task*, *entry*)

flexget.plugins.urlrewrite_btjunkie.**register_plugin**()

3.1.24 urlrewrite_deadfrog Module

class flexget.plugins.urlrewrite_deadfrog.**UrlRewriteDeadFrog**
Bases: object

DeadFrog urlrewriter.

parse_download_page(**args*, ***kwargs*)

url_rewritable(*task*, *entry*)

url_rewrite(*task*, *entry*)

flexget.plugins.urlrewrite_deadfrog.**register_plugin**()

3.1.25 urlrewrite_extratorrent Module

class flexget.plugins.urlrewrite_extratorrent.**UrlRewriteExtraTorrent**
Bases: object

ExtraTorrent search plugin.

should accept: kat:

category: <category>

categories: all music anime adult movies tv

schema = {'additionalProperties': False, 'type': 'object', 'properties': {'category': {'enum': ['all', 'music', 'anime', 'adu

search(*task*, *entry*, *config=None*)

url_rewritable(*task*, *entry*)

url_rewrite(*task*, *entry*)

flexget.plugins.urlrewrite_extratorrent.**register_plugin**()

3.1.26 urlrewrite_google_cse Module

class flexget.plugins.urlrewrite_google_cse.**UrlRewriteGoogle**
Bases: object

url_rewritable(*task*, *entry*)

url_rewrite(*task*, *entry*)

class flexget.plugins.urlrewrite_google_cse.**UrlRewriteGoogleCse**
Bases: object

Google custom query urlrewriter.

url_rewritable(*task*, *entry*)

url_rewrite(*task*, *entry*)

flexget.plugins.urlrewrite_google_cse.**register_plugin**()

3.1.27 urlrewrite_isohunt Module

class flexget.plugins.urlrewrite_isohunt.**UrlRewriteIsoHunt**
Bases: object

IsoHunt urlrewriter and search plugin.

should accept: isohunt: <category>

categories: empty or -1: All 0 : Misc. 1 : Video/Movies 2 : Audio 3 : TV 4 : Games 5 : Apps 6 : Pics 7 : Anime 8 : Comics 9 : Books 10: Music Video 11: Unclassified 12: ALL

schema = {u'enum': [u'misc', u'movies', u'audio', u'tv', u'games', u'apps', u'pics', u'anime', u'comics', u'books', u'mus

search(*task*, *entry*, *config*)

url_rewritable(*task*, *entry*)

url_rewrite(*task*, *entry*)

flexget.plugins.urlrewrite_isohunt.**register_plugin**()

3.1.28 urlrewrite_newtorrents Module

class flexget.plugins.urlrewrite_newtorrents.**NewTorrents**
NewTorrents urlrewriter and search plugin.

entries_from_search(**args*, ***kwargs*)

search(*task*, *entry*, *config=None*)

url_from_page(**args*, ***kwargs*)

url_rewritable(*task*, *entry*)

url_rewrite(*task*, *entry*)

flexget.plugins.urlrewrite_newtorrents.**register_plugin**()

3.1.29 urlrewrite_newzleech Module

class flexget.plugins.urlrewrite_newzleech.**UrlRewriteNewzleech**
Bases: object

UrlRewriter or search by using newzleech.com TODO: implement basic url rewriting

search(**args*, ***kwargs*)

flexget.plugins.urlrewrite_newzleech.**register_plugin**()

3.1.30 urlrewrite_nyaa Module

class flexget.plugins.urlrewrite_nyaa.**UrlRewriteNyaa**
Bases: object

Nyaa urlrewriter and search plugin.

search(*task*, *entry*, *config*)

url_rewritable(*task*, *entry*)

url_rewrite(*task*, *entry*)

validator()

flexget.plugins.urlrewrite_nyaa.**register_plugin**()

3.1.31 urlrewrite_piratebay Module

class flexget.plugins.urlrewrite_piratebay.**UrlRewritePirateBay**
Bases: object

PirateBay urlrewriter.

parse_download_page(**args*, ***kwargs*)

schema = {u'oneOf': [{u'type': u'boolean'}, {u'additionalProperties': False, u'type': u'object', u'properties': {u'categor

search(**args*, ***kwargs*)

url_rewritable(*task*, *entry*)

url_rewrite(*task*, *entry*)

flexget.plugins.urlrewrite_piratebay.**register_plugin**()

3.1.32 urlrewrite_redskunk Module

class flexget.plugins.urlrewrite_redskunk.**UrlRewriteRedskunk**
Bases: object

Redskunk urlrewriter.

url_rewritable(*task*, *entry*)

url_rewrite(*task*, *entry*)

flexget.plugins.urlrewrite_redskunk.**register_plugin**()

3.1.33 urlrewrite_search Module

class flexget.plugins.urlrewrite_search.**PluginSearch**
Bases: object

Search entry from sites. Accepts list of known search plugins, list is in priority order. Once hit has been found no more searches are performed. Should be used only when there is no other way to get working download url, ie. when input plugin does not provide any downloadable urls.

Example:

```
urlrewrite_search:
  - newtorrents
  - piratebay
```

Note: Some url rewriters will use search plugins automatically if entry url points into a search page.

on_task_urlrewrite(*task*, *config*)

schema = {u'items': {u'allOf': [{u'$ref': u'/schema/plugins?group=search'}, {u'maxProperties': 1, u'minProperties': 1}

flexget.plugins.urlrewrite_search.**register_plugin**()

3.1.34 urlrewrite_stmusic Module

class flexget.plugins.urlrewrite_stmusic.**UrlRewriteSTMusic**
Bases: object

STMusic urlrewriter.

url_rewritable(*task*, *entry*)

url_rewrite(*task*, *entry*)

flexget.plugins.urlrewrite_stmusic.**register_plugin**()

3.1.35 urlrewrite_torrentz Module

class flexget.plugins.urlrewrite_torrentz.**UrlRewriteTorrentz**
Bases: object

Torrentz urlrewriter.

process_config(*config*)
Return plugin configuration in advanced form

schema = {u'oneOf': [{u'additionalProperties': False, u'type': u'object', u'properties': {u'extra_terms': {u'type': u'stri

search(*task*, *entry*, *config=None*)

url_rewritable(*task*, *entry*)

url_rewrite(*task*, *entry*)

flexget.plugins.urlrewrite_torrentz.**register_plugin**()

3.1.36 urlrewrite_urlrewrite Module

class flexget.plugins.urlrewrite_urlrewrite.**UrlRewrite**
Bases: object

Generic configurable urlrewriter.

Example:

```
urlrewrite:
  demonoid:
    regexp: http://www\.demonoid\.com/files/details/
    format: http://www.demonoid.com/files/download/HTTP/
```

on_task_start(*task*, *config*)

resolves = {}

schema = {u'additionalProperties': {u'additionalProperties': False, u'required': [u'regexp', u'format'], u'type': u'objec

url_rewritable(*task*, *entry*)

url_rewrite(*task*, *entry*)

flexget.plugins.urlrewrite_urlrewrite.**register_plugin**()

3.1.37 Subpackages

cli Package

cli Package

cli_config Module

flexget.plugins.cli.cli_config.**key_value_pair**(*text*)

flexget.plugins.cli.cli_config.**log** = <flexget.logger.FlexGetLogger object>
Allows specifying yml configuration values from commandline parameters.

Yml variables are prefixed with dollar sign ($). Commandline parameter must be comma separated list of variable=values.

Configuration example:

```
tasks:
  my task:
    rss: $url
    download: $path
```

Commandline example:

```
--cli-config url=http://some.url/ path=~/downloads
```

flexget.plugins.cli.cli_config.**register_parser_arguments**()

flexget.plugins.cli.cli_config.**replace_in_item**(*replaces*, *item*)

flexget.plugins.cli.cli_config.**substitute_cli_variables**(*config*, *manager*)

doc Module

flexget.plugins.cli.doc.**print_doc**(*manager*, *options*)

flexget.plugins.cli.doc.**register_parser_arguments**()

flexget.plugins.cli.doc.**trim**(*docstring*)

explain_sql Module

class flexget.plugins.cli.explain_sql.**Explain**(*stmt*)
Bases: sqlalchemy.sql.base.Executable, sqlalchemy.sql.elements.ClauseElement

class flexget.plugins.cli.explain_sql.**ExplainQuery**(*entities*, *session=None*)
Bases: sqlalchemy.orm.query.Query

flexget.plugins.cli.explain_sql.**deregister_sql_explain**(*man*, *options*)

flexget.plugins.cli.explain_sql.**explain**(*element*, *compiler*, ***kw*)

flexget.plugins.cli.explain_sql.**register_parser_arguments**()

flexget.plugins.cli.explain_sql.**register_sql_explain**(*man*, *options*)

movie_queue Module

flexget.plugins.cli.movie_queue.**clear**()
Deletes waiting movies from queue

flexget.plugins.cli.movie_queue.**do_cli**(*manager*, *options*)
Handle movie-queue subcommand

flexget.plugins.cli.movie_queue.**queue_list**(*options*)
List movie queue

flexget.plugins.cli.movie_queue.**register_parser_arguments**()

perf_tests Module

flexget.plugins.cli.perf_tests.**cli_perf_test**(*manager*, *options*)

flexget.plugins.cli.perf_tests.**imdb_query**(*session*)

flexget.plugins.cli.perf_tests.**register_parser_arguments**()

performance Module

flexget.plugins.cli.performance.**after_plugin**(*task*, *keyword*)

flexget.plugins.cli.performance.**before_plugin**(*task*, *keyword*)

flexget.plugins.cli.performance.**cleanup**(*manager*, *options*)

flexget.plugins.cli.performance.**log_query_count**(*name_point*)
Debugging purposes, allows logging number of executed queries at :name_point:

flexget.plugins.cli.performance.**register_parser_arguments**()

flexget.plugins.cli.performance.**startup**(*manager*, *options*)

plugins Module

flexget.plugins.cli.plugins.**plugins_summary**(*manager*, *options*)

flexget.plugins.cli.plugins.**register_parser_arguments**()

series Module

flexget.plugins.cli.series.**begin**(*manager*, *options*)

flexget.plugins.cli.series.**display_details**(*name*)
Display detailed series information, ie. series show NAME

flexget.plugins.cli.series.**display_summary**(*options*)
Display series summary. :param options: argparse options from the CLI

flexget.plugins.cli.series.**do_cli**(*manager*, *options*)

flexget.plugins.cli.series.**forget**(*manager*, *options*)

flexget.plugins.cli.series.**get_latest_status**(*episode*)

Parameters **episode** – Instance of Episode

Returns Status string for given episode

flexget.plugins.cli.series.**register_parser_arguments**()

download Package

download Package

Plugins for "download" task phase.

exit Package

exit Package

Plugins for "exit" task phase.

filter Package

filter Package

Plugins for "filter" task phase, and non-modifying download filters.

accept_all Module

class flexget.plugins.filter.accept_all.**FilterAcceptAll**
Bases: object

Just accepts all entries.

Example:

```
accept_all: true
```

on_task_filter(*task*, *config*)

schema = {u'type': u'boolean'}

flexget.plugins.filter.accept_all.**register_plugin**()

all_series Module

class flexget.plugins.filter.all_series.**FilterAllSeries**
Bases: *flexget.plugins.filter.series.FilterSeriesBase*

Grabs all entries that appear to be series episodes in a task.

This plugin just configures the series plugin dynamically with all series from the task. It can take any of the options of the series plugin.

Example:: all_series: yes

::

all_series: quality: hdtv+ propers: no

on_task_metainfo(*task*, *config*)

schema

flexget.plugins.filter.all_series.**register_plugin**()

content_filter Module

class flexget.plugins.filter.content_filter.**FilterContentFilter**
Bases: object

Rejects entries based on the filenames in the content. Torrent files only right now.

Example:

```
content_filter:
  require:
    - '*.avi'
    - '*.mkv'
```

on_task_modify(*task*, *config*)

prepare_config(*config*)

process_entry(*task*, *entry*, *config*)
Process an entry and reject it if it doesn't pass filter.

Parameters

- **task** – Task entry belongs to.
- **entry** – Entry to process

Returns True, if entry was rejected.

schema = {u'additionalProperties': False, u'type': u'object', u'properties': {u'require_mainfile': {u'default': False, u'ty

flexget.plugins.filter.content_filter.**register_plugin**()

content_size Module

class flexget.plugins.filter.content_size.**FilterContentSize**
Bases: object

on_task_filter(*task*, *config*)

on_task_modify(*task*, *config*)

process_entry(*task*, *entry*, *config*, *remember=True*)
Rejects this entry if it does not pass content_size requirements. Returns true if the entry was rejected.

schema = {u'additionalProperties': False, u'type': u'object', u'properties': {u'max': {u'type': u'number'}, u'strict': {u'

flexget.plugins.filter.content_size.**register_plugin**()

crossmatch Module

class flexget.plugins.filter.crossmatch.**CrossMatch**
Bases: object

Perform action based on item on current task and other inputs.

Example:

```
crossmatch:
  from:
    - rss: http://example.com/
  fields:
    - title
  action: reject
```

entry_intersects(*e1*, *e2*, *fields=None*)

Parameters

- **e1** – First *flexget.entry.Entry*
- **e2** – Second *flexget.entry.Entry*
- **fields** – List of fields which are checked

Returns List of field names in common

on_task_filter(*task*, *config*)

schema = {u'additionalProperties': False, u'required': [u'fields', u'action', u'from'], u'type': u'object', u'properties': {u

flexget.plugins.filter.crossmatch.**register_plugin**()

delay Module

class flexget.plugins.filter.delay.**DelayedEntry**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

entry

expire

id

task

title

class flexget.plugins.filter.delay.**FilterDelay**
Bases: object

Add delay to a task. This is useful for de-prioritizing expensive / bad-quality tasks.

Format: n [minutes|hours|days|weeks]

Example:

```
delay: 2 hours
```

get_delay(*config*)

on_task_input(*task*, *config*)
Captures the current input then replaces it with entries that have passed the delay.

schema = {u'type': u'string', u'format': u'interval'}

flexget.plugins.filter.delay.**register_plugin**()

exists Module

class flexget.plugins.filter.exists.**FilterExists**
Bases: object

Reject entries that already exist in given path.

Example:

```
exists: /storage/movies/
```

on_task_filter(*task*, *config*)

prepare_config(*config*)

schema = {u'oneOf': [{u'minItems': 1, u'items': {u'title': u'single value', u'type': u'string', u'format': u'path'}, u'type'

flexget.plugins.filter.exists.**register_plugin**()

exists_movie Module

class flexget.plugins.filter.exists_movie.**FilterExistsMovie**
Bases: object

Reject existing movies.

Syntax:

exists_movie: path: /path/to/movies [type: {dirs|files}] [allow_different_qualities: {better|yes|no}] [lookup: {imdb|no}]

dir_pattern = <_sre.SRE_Pattern object>

file_pattern = <_sre.SRE_Pattern object>

on_task_filter(*task*, *config*)

prepare_config(*config*)

schema = {u'anyOf': [{u'oneOf': [{u'minItems': 1, u'items': {u'title': u'single value', u'type': u'string', u'format': u'pa

flexget.plugins.filter.exists_movie.**register_plugin**()

exists_series Module

class flexget.plugins.filter.exists_series.**FilterExistsSeries**
Bases: object

Intelligent series aware exists rejecting.

Example:

```
exists_series: /storage/series/
```

on_task_filter(*task*, *config*)

prepare_config(*config*)

schema = {u'anyOf': [{u'oneOf': [{u'minItems': 1, u'items': {u'title': u'single value', u'type': u'string', u'format': u'pa

flexget.plugins.filter.exists_series.**register_plugin**()

if_condition Module

class flexget.plugins.filter.if_condition.**FilterIf**
Bases: object

Can run actions on entries that satisfy a given condition.

Actions include accept, reject, and fail, as well as the ability to run other filter plugins on the entries.

check_condition(*condition*, *entry*)
Checks if a given *entry* passes *condition*

schema = {u'items': {u'additionalProperties': {u'anyOf': [{u'$ref': u'/schema/plugins'}, {u'enum': [u'accept', u'reject',

flexget.plugins.filter.if_condition.**register_plugin**()

flexget.plugins.filter.if_condition.**safer_eval**(*statement*, *locals*)
A safer eval function. Does not allow __ or try statements, only includes certain 'safe' builtins.

imdb Module

class flexget.plugins.filter.imdb.**FilterImdb**
Bases: object

This plugin allows filtering based on IMDB score, votes and genres etc.

Note: All parameters are optional. Some are mutually exclusive.

Configuration:

```
min_score: <num>
min_votes: <num>
min_year: <num>
max_year: <num>

# accept movies with any of these genres
accept_genres:
  - genre1
  - genre2

# reject if genre contains any of these
reject_genres:
  - genre1
  - genre2

# reject if language contain any of these
reject_languages:
  - language1

# accept only these primary languages
accept_languages:
  - language1

# accept movies with any of these actors
accept_actors:
  - nm0004695
  - nm0004754

# reject movie if it has any of these actors
reject_actors:
  - nm0001191
  - nm0002071

# accept all movies by these directors
accept_directors:
  - nm0000318

# reject movies by these directors
reject_directors:
  - nm0093051

# reject movies/TV shows with any of these ratings
reject_mpaa_ratings:
  - PG_13
  - R
  - X

# accept movies/TV shows with only these ratings
accept_mpaa_ratings:
  - PG
  - G
  - TV_Y
```

on_task_filter(*task*, *config*)

schema = {u'additionalProperties': False, u'type': u'object', u'properties': {u'min_votes': {u'type': u'integer'}, u'reject

flexget.plugins.filter.imdb.**register_plugin**()

imdb_rated Module

imdb_required Module

class flexget.plugins.filter.imdb_required.**FilterImdbRequired**
Bases: object

Rejects entries without imdb_url or imdb_id. Makes imdb lookup / search if necessary.

Example:

```
imdb_required: yes
```

on_task_filter(*task*, *config*)

schema = {u'type': u'boolean'}

flexget.plugins.filter.imdb_required.**register_plugin**()

limit_new Module

class flexget.plugins.filter.limit_new.**FilterLimitNew**
Bases: object

Limit number of new items.

Example:

```
limit_new: 1
```

This would allow only one new item to pass trough per execution. Useful for passing torrents slowly into download.

Note that since this is per execution, actual rate depends how often FlexGet is executed.

on_task_filter(*task*, *config*)

schema = {u'minimum': 1, u'type': u'integer'}

flexget.plugins.filter.limit_new.**register_plugin**()

movie_queue Module

class flexget.plugins.filter.movie_queue.**MovieQueue**
Bases: *flexget.plugins.filter.queue_base.FilterQueueBase*

matches(*task*, *config*, *entry*)

on_task_output(*task*, *config*)

schema = {u'oneOf': [{u'enum': [u'accept', u'add', u'remove', u'forget'], u'type': u'string'}, {u'additionalProperties': F

exception flexget.plugins.filter.movie_queue.**QueueError**(*message*, *errno=0*)
Bases: exceptions.Exception

Exception raised if there is an error with a queue operation

class flexget.plugins.filter.movie_queue.**QueuedMovie**(***kwargs*)
Bases: *flexget.plugins.filter.queue_base.QueuedItem*

added

discriminator

downloaded

entry_original_url

entry_title

entry_url

id

imdb_id

quality

title

tmdb_id

flexget.plugins.filter.movie_queue.**migrate_imdb_queue**(*manager*)
If imdb_queue table is found, migrate the data to movie_queue

flexget.plugins.filter.movie_queue.**register_plugin**()

only_new Module

class flexget.plugins.filter.only_new.**FilterOnlyNew**
Bases: object

Causes input plugins to only emit entries that haven't been seen on previous runs.

on_task_learn(*task*, *config*)
Reject all entries so remember_rejected will reject them next time

on_task_start(*task*, *config*)
Make sure the remember_rejected plugin is available

schema = {u'type': u'boolean'}

flexget.plugins.filter.only_new.**register_plugin**()

private_torrents Module

class flexget.plugins.filter.private_torrents.**FilterPrivateTorrents**
Bases: object

How to handle private torrents.

private_torrents: yes|no

Example:

```
private_torrents: no
```

This would reject all torrent entries with private flag.

Example:

```
private_torrents: yes
```

This would reject all public torrents.

Non-torrent content is not interviened.

on_task_modify(*task*, *config*)

schema = {u'type': u'boolean'}

flexget.plugins.filter.private_torrents.**register_plugin**()

proper_movies Module

class flexget.plugins.filter.proper_movies.**FilterProperMovies**
Bases: object

Automatically download proper movies.

Configuration:

proper_movies: n <minutes|hours|days|weeks>

or permanently:

proper_movies: yes

Value no will disable plugin.

on_task_filter(*task*, *config*)

on_task_learn(*task*, *config*)
Add downloaded movies to the database

schema = {u'oneOf': [{u'type': u'boolean'}, {u'type': u'string', u'format': u'interval'}]}

class flexget.plugins.filter.proper_movies.**ProperMovie**
Bases: sqlalchemy.ext.declarative.api.Base

added

id

imdb_id

proper_count

quality

task

title

flexget.plugins.filter.proper_movies.**register_plugin**()

quality Module

class flexget.plugins.filter.quality.**FilterQuality**
Bases: object

Rejects all entries that don't have one of the specified qualities

Example:

```
quality:
  - hdtv
```

on_task_filter(*task*, *config*)

schema = {u'oneOf': [{u'minItems': 1, u'items': {u'title': u'single value', u'type': u'string', u'format': u'quality_requir

flexget.plugins.filter.quality.**register_plugin**()

queue_base Module

class flexget.plugins.filter.queue_base.**FilterQueueBase**
Bases: object

Base class to handle general tasks of keeping a queue of wanted items.

matches(*task*, *config*, *entry*)
This should return the QueueItem object for the match, if this entry is in the queue.

on_task_filter(*task*, *config*)

on_task_learn(*task*, *config*)

on_task_start(*task*, *config*)

schema = {u'type': u'boolean'}

class flexget.plugins.filter.queue_base.**QueuedItem**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

added

discriminator

downloaded

entry_original_url

entry_title

entry_url

id

title

regexp Module

class flexget.plugins.filter.regexp.**FilterRegexp**
Bases: object

All possible forms.

regexp:

[operation]: # operation to perform on matches

- [regexp] # simple regexp
- [regexp]: <path> # override path
- **[regexp]:** [path]: <path> # override path [not]: <regexp> # not match [from]: <field> # search from given entry field
- **[regexp]:** [path]: <path> # override path [not]: # list of not match regexps
  - <regexp>

  [from]: # search only from these fields
  - <field>

[operation]:

- <regexp>

[rest]: <operation> # non matching entries are [from]: # search only from these fields for all regexps

- <field>

Possible operations: accept, reject, accept_excluding, reject_excluding

filter(*task*, *operation*, *regexps*)

Parameters

- **task** – Task instance
- **operation** – one of 'accept' 'reject' 'accept_excluding' and 'reject_excluding' accept and reject will be called on the entry if any of the regxps match *_excluding operations will be called if any of the regexps don't match
- **regexps** – list of {compiled_regexp: options} dictionaries

Returns Return list of entries that didn't match regexps

matches(*entry*, *regexp*, *find_from=None*, *not_regexps=None*)

Check if :entry: has any string fields or strings in a list field that match :regexp:

Parameters

- **entry** – Entry instance
- **regexp** – Compiled regexp
- **find_from** – None or a list of fields to search from
- **not_regexps** – None or list of regexps that can NOT match

Returns Field matching

on_task_filter(*task*, *config*)

prepare_config(*config*)

Returns the config in standard format.

All regexps are turned into dictionaries in the form of {compiled regexp: options}

Parameters **config** – Dict that can optionally contain the following keys path: will be attached to entries that match set: a dict of values to be attached to entries that match via set plugin from: a list of fields in entry for the regexps to match against not: a list of compiled regexps that if matching, will disqualify the main match

Returns New config dictionary

schema = {u'additionalProperties': False, u'definitions': {u'regex_list': {u'items': {u'oneOf': [{u'type': u'string', u'forn

flexget.plugins.filter.regexp.**register_plugin**()

retry_failed Module

class flexget.plugins.filter.retry_failed.**FailedEntry**(*title*, *url*, *reason=None*)

Bases: sqlalchemy.ext.declarative.api.Base

count

id

reason

retry_time

title

tof

url

class flexget.plugins.filter.retry_failed.**PluginFailed**
Bases: object

Records entry failures and stores them for trying again after a certain interval. Rejects them after they have failed too many times.

add_failed(*entry*, *reason=None*, *config=None*, ***kwargs*)
Adds entry to internal failed list, displayed with –failed

on_task_filter(*task*, *config*)

on_task_input(*task*, *config*)

prepare_config(*config*)

retry_time(*fail_count*, *config*)
Return the timedelta an entry that has failed *fail_count* times before should wait before being retried.

schema = {u'oneOf': [{u'type': u'boolean'}, {u'additionalProperties': False, u'type': u'object', u'properties': {u'retry_t

flexget.plugins.filter.retry_failed.**clear_failed**(*manager*)

flexget.plugins.filter.retry_failed.**do_cli**(*manager*, *options*)

flexget.plugins.filter.retry_failed.**list_failed**()

flexget.plugins.filter.retry_failed.**register_parser_arguments**()

flexget.plugins.filter.retry_failed.**register_plugin**()

remember_rejected Module

class flexget.plugins.filter.remember_rejected.**FilterRememberRejected**
Bases: object

Internal. Rejects entries which have been rejected in the past.

This is enabled when item is rejected with remember=True flag.

Example:: entry.reject('message', remember=True)

on_entry_reject(*entry*, *remember=None*, *remember_time=None*, ***kwargs*)

on_task_filter(*task*, *config*)
Reject any remembered entries from previous runs

on_task_input(*task*, *config*)

on_task_learn(*task*, *config*)

on_task_start(*task*, *config*)
Purge remembered entries if the config has changed.

class flexget.plugins.filter.remember_rejected.**RememberEntry**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

added

expires

id

reason

rejected_by

task_id

title

url

class flexget.plugins.filter.remember_rejected.**RememberTask**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

entries

id

name

flexget.plugins.filter.remember_rejected.**clear_rejected**(*manager*)

flexget.plugins.filter.remember_rejected.**db_cleanup**(*manager*, *session*)

flexget.plugins.filter.remember_rejected.**do_cli**(*manager*, *options*)

flexget.plugins.filter.remember_rejected.**list_rejected**()

flexget.plugins.filter.remember_rejected.**register_parser_arguments**()

flexget.plugins.filter.remember_rejected.**register_plugin**()

require_field Module

class flexget.plugins.filter.require_field.**FilterRequireField**
Bases: object

Rejects entries without defined field.

Example:

```
require_field: imdb_url
```

on_task_filter(*task*, *config*)

schema = {u'oneOf': [{u'minItems': 1, u'items': {u'type': u'string', u'title': u'single value'}, u'type': u'array', u'title': u

flexget.plugins.filter.require_field.**register_plugin**()

seen Module

Listens events:

forget (string)

Given string can be task name, remembered field (url, imdb_url) or a title. If given value is a task name then everything in that task will be forgotten. With title all learned fields from it and the title will be forgotten. With field value only that particular field is forgotten.

class flexget.plugins.filter.seen.**FilterSeen**
Bases: object

Remembers previously downloaded content and rejects them in subsequent executions. Without this plugin FlexGet would download all matching content on every execution.

This plugin is enabled on all tasks by default. See wiki for more information.

forget(*task*, *title*)
Forget SeenEntry with :title:. Return True if forgotten.

learn(*task*, *entry*, *fields=None*, *reason=None*, *local=False*)
Marks entry as seen

on_task_filter(*task*, *config*, *remember_rejected=False*)
Filter entries already accepted on previous runs.

on_task_learn(*task*, *config*)
Remember succeeded entries

schema = {u'oneOf': [{u'type': u'boolean'}, {u'enum': [u'global', u'local'], u'type': u'string'}]}

class flexget.plugins.filter.seen.**SeenEntry**(*title*, *task*, *reason=None*, *local=False*)
Bases: sqlalchemy.ext.declarative.api.Base

added

fields

id

local

reason

task

title

class flexget.plugins.filter.seen.**SeenField**(*field*, *value*)
Bases: sqlalchemy.ext.declarative.api.Base

added

field

id

seen_entry_id

value

flexget.plugins.filter.seen.**db_cleanup**(*manager*, *session*)

flexget.plugins.filter.seen.**do_cli**(*manager*, *options*)

flexget.plugins.filter.seen.**forget**(*value*)
See module docstring :param string value: Can be task name, entry title or field value :return: count, field_count where count is number of entries removed and field_count number of fields

flexget.plugins.filter.seen.**register_parser_arguments**()

flexget.plugins.filter.seen.**register_plugin**()

flexget.plugins.filter.seen.**seen_add**(*options*)

flexget.plugins.filter.seen.**seen_forget**(*manager*, *options*)

flexget.plugins.filter.seen.**seen_search**(*options*)

seen_info_hash Module

class flexget.plugins.filter.seen_info_hash.**FilterSeenInfoHash**
Bases: *flexget.plugins.filter.seen.FilterSeen*

Prevents the same torrent from being downloaded twice by remembering the infohash of all downloaded torrents.

on_task_filter(*task*, *config*)

on_task_modify(*task*, *config*)

schema = {u'oneOf': [{u'type': u'boolean'}, {u'enum': [u'global', u'local'], u'type': u'string'}]}

flexget.plugins.filter.seen_info_hash.**register_plugin**()

seen_movies Module

class flexget.plugins.filter.seen_movies.**FilterSeenMovies**
Bases: *flexget.plugins.filter.seen.FilterSeen*

Prevents movies being downloaded twice. Works only on entries which have imdb url available.

How duplicate movie detection works: 1) Remember all imdb urls from downloaded entries. 2) If stored imdb url appears again, entry is rejected.

on_task_filter(*task*, *config*)

schema = {u'oneOf': [{u'enum': [u'strict', u'loose'], u'type': u'string'}, {u'additionalProperties': False, u'type': u'objec

flexget.plugins.filter.seen_movies.**register_plugin**()

series Module

class flexget.plugins.filter.series.**AlternateNames**(*name*)
Bases: sqlalchemy.ext.declarative.api.Base

Similar to Series. Name is handled case insensitively transparently.

alt_name

id

name_comparator()

name_getter()

name_setter(*value*)

series_id

class flexget.plugins.filter.series.**Episode**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

age

Returns Pretty string representing age of episode. eg "23d 12h" or "No releases seen"

downloaded_releases

first_seen = <sqlalchemy.sql.elements.Label object>

id

identified_by

identifier

is_premiere

number

releases

season

series_id

class flexget.plugins.filter.series.**FilterSeries**
Bases: *flexget.plugins.filter.series.FilterSeriesBase*

Intelligent filter for tv-series.

http://flexget.com/wiki/Plugins/series

auto_exact(*config*)
Automatically enable exact naming option for series that look like a problem

on_task_filter(*task*, *config*)
Filter series

on_task_learn(*task*, *config*)
Learn succeeded episodes

on_task_metainfo(*task*, *config*)

parse_series(*entries*, *series_name*, *config*)
Search for *series_name* and populate all *series_** fields in entries when successfully parsed

Parameters

- **session** – SQLAlchemy session
- **entries** – List of entries to process
- **series_name** – Series name which is being processed
- **config** – Series config being processed

process_episode_tracking(*episode*, *entries*, *grace*, *backfill=False*)
Rejects all episodes that are too old or new, return True when this happens.

Parameters

- **episode** – Episode model
- **entries** (*list*) – List of entries for given episode.
- **grace** (*int*) – Number of episodes before or after latest download that are allowed.
- **backfill** (*bool*) – If this is True, previous episodes will be allowed, but forward advancement will still be restricted.

process_propers(*config*, *episode*, *entries*)
Accepts needed propers. Nukes episodes from which there exists proper.

Returns A list of episodes to continue processing.

process_qualities(*config*, *entries*, *downloaded*)
Handles all modes that can accept more than one quality per episode. (qualities, upgrade)

Returns True - if at least one wanted quality has been downloaded or accepted. False - if no wanted qualities have been accepted

process_quality(*config*, *entries*)
Filters eps that do not fall between within our defined quality standards.

Returns A list of eps that are in the acceptable range

process_series(*task*, *series_entries*, *config*)
Accept or Reject episode from available releases, or postpone choosing.

Parameters

- **task** – Current Task
- **series_entries** – dict mapping Episodes to entries for that episode
- **config** – Series configuration

process_timeframe(*task*, *config*, *episode*, *entries*)
Runs the timeframe logic to determine if we should wait for a better quality. Saves current best to backlog if timeframe has not expired.

Returns True - if we should keep the quality (or qualities) restriction False - if the quality restriction should be released, due to timeframe expiring

process_timeframe_target(*config*, *entries*, *downloaded=None*)
Accepts first episode matching the quality configured for the series.

Returns True if accepted something

schema

class flexget.plugins.filter.series.**FilterSeriesBase**
Bases: object

Class that contains helper methods for both filter.series as well as plugins that configure it, such as all_series, series_premiere and configure_series.

apply_group_options(*config*)
Applies group settings to each item in series group and removes settings dict.

combine_series_lists(**series_lists*, ***kwargs*)
Combines the series from multiple lists, making sure there are no doubles.

If keyword argument log_once is set to True, an error message will be printed if a series is listed more than once, otherwise log_once will be used.

make_grouped_config(*config*)
Turns a simple series list into grouped format with a empty settings dict

merge_config(*task*, *config*)
Merges another series config dict in with the current one.

prepare_config(*config*)
Generate a list of unique series from configuration. This way we don't need to handle two different configuration formats in the logic. Applies group settings with advanced form.

settings_schema

class flexget.plugins.filter.series.**NormalizedComparator**(*expression*)
Bases: sqlalchemy.ext.hybrid.Comparator

operate(*op*, *other*)

class flexget.plugins.filter.series.**Release**
Bases: sqlalchemy.ext.declarative.api.Base

downloaded

episode_id

first_seen

id

proper

proper_count

quality

title

class flexget.plugins.filter.series.**Series**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Name is handled case insensitively transparently

alternate_names

begin

begin_episode_id

episodes

id

identified_by

in_tasks

name

name_comparator()

name_getter()

name_setter(*value*)

class flexget.plugins.filter.series.**SeriesDBManager**
Bases: *flexget.plugins.filter.series.FilterSeriesBase*

Update in the database with series info from the config

on_task_start(*task*, *config*)

class flexget.plugins.filter.series.**SeriesTask**(*name*)
Bases: sqlalchemy.ext.declarative.api.Base

id

name

series_id

flexget.plugins.filter.series.**auto_identified_by**(*series*)
Determine if series *name* should be considered identified by episode or id format

Returns 'ep', 'sequence', 'date' or 'id' if enough history is present to identify the series' id type. Returns 'auto' if there is not enough history to determine the format yet

flexget.plugins.filter.series.**clean_series**(*manager*)

flexget.plugins.filter.series.**db_cleanup**(*manager*, *session*)

flexget.plugins.filter.series.**forget_series**(*name*)
Remove a whole series *name* from database.

flexget.plugins.filter.series.**forget_series_episode**(*name*, *identifier*)
Remove all episodes by *identifier* from series *name* from database.

flexget.plugins.filter.series.**get_latest_episode**(*series*)
Return latest known identifier in dict (season, episode, name) for series name

flexget.plugins.filter.series.**get_latest_release**(*series*, *downloaded=True*, *season=None*)

Parameters

- **series** (Series) – SQLAlchemy session
- **Downloaded** – find only downloaded releases
- **Season** – season to find newest release for

Returns Instance of Episode or None if not found.

flexget.plugins.filter.series.**new_eps_after**(*since_ep*)

Parameters **since_ep** – Episode instance

Returns Number of episodes since then

flexget.plugins.filter.series.**normalize_series_name**(*name*)
Returns a normalized version of the series name.

flexget.plugins.filter.series.**populate_entry_fields**(*entry*, *parser*, *config*)
Populates all series_fields for given entry based on parser.

Parameters **parser** – A valid result from a series parser used to populate the fields.

Config dict If supplied, will use 'path' and 'set' options to populate specified fields.

flexget.plugins.filter.series.**register_parser_arguments**()

flexget.plugins.filter.series.**register_plugin**()

flexget.plugins.filter.series.**repair**(*manager*)

flexget.plugins.filter.series.**set_series_begin**(*series*, *ep_id*)
Set beginning for series

Parameters

- **series** (Series) – Series instance
- **ep_id** – Integer for sequence mode, SxxEyy for episodic and yyyy-mm-dd for date.

Raises ValueError If malformed ep_id or series in different mode

flexget.plugins.filter.series.**store_parser**(*session*, *parser*, *series=None*, *quality=None*)
Push series information into database. Returns added/existing release.

Parameters

- **session** – Database session to use
- **parser** – parser for release that should be added to database
- **series** – Series in database to add release to. Will be looked up if not provided.

- **quality** – If supplied, this will override the quality from the series parser

Returns List of Releases

series_premiere Module

class flexget.plugins.filter.series_premiere.**FilterSeriesPremiere**
Bases: *flexget.plugins.filter.series.FilterSeriesBase*

Accept an entry that appears to be the first episode of any series.

Can be configured with any of the options of series plugin Examples:

series_premiere: yes

series_premiere: path: ~/Media/TV/_NEW_/. quality: 720p timeframe: 12 hours

NOTE: this plugin only looks in the entry title and expects the title format to start with the series name followed by the episode info. Use the manipulate plugin to modify the entry title to match this format, if necessary.

TODO:

- integrate thetvdb to allow refining by genres, etc.

on_task_metainfo(*task*, *config*)

schema

flexget.plugins.filter.series_premiere.**register_plugin**()

thetvdb Module

class flexget.plugins.filter.thetvdb.**FilterTvdb**
Bases: object

This plugin allows filtering based on thetvdb series rating, episode rating, status, genres, runtime, content-rating, languages, directors, writers, network, guest stars, episode rating, and actors

Configuration:

Note: All parameters are optional. Some are mutually exclusive.

min_series_rating: <num> min_episode_rating: <num> min_episode_air_year: <num> max_episode_air_year: <num> min_episode_runtime: <num> max_episode_runtime: <num>

reject if genre contains any of these reject_content_rating:

•TV-MA

accept only this content rating accept_content_rating:

•TV-PG

accept only these networks accept_network:

•NBC

reject if this network reject_network:

•ABC

reject if genre contains any of these reject_genres:

•drama

•romance

reject if status contains any of these reject_status:

•Ended

reject if language contain any of these reject_languages:

•fr

accept only this language accept_languages:

•en

Actors below take into account series actors, and guest stars # accept episode with any of these actors accept_actors:

•Heidi Klum

•Bruce Willis

reject episode if it has any of these actors reject_actors:

•Cher

•Tamala Jones

accept all episodes by these writers accept_writers:

•Andrew W. Marlowe

reject episodes by these writers reject_writers:

•Barry Schindel

accept all episodes by these directors accept_directors:

•Rob Bowman

reject movies by these directors reject_directors:

•John Terlesky

is_in_set(*config*, *configkey*, *entryitem*)
this takes the config object, config key (to a list), and entry item so it can return True if the object matches, (be that a subset of the list, or if the entry item is contained within the config object list) or false if it does not.

on_task_filter(*task*, *config*)

schema = {u'additionalProperties': False, u'type': u'object', u'properties': {u'reject_languages': {u'items': {u'type': u's

flexget.plugins.filter.thetvdb.**register_plugin**()

torrent_alive Module

class flexget.plugins.filter.torrent_alive.**TorrentAlive**
Bases: object

on_task_filter(*task*, *config*)

on_task_output(*task*, *config*)

prepare_config(*config*)

schema = {u'oneOf': [{u'type': u'boolean'}, {u'type': u'integer'}, {u'additionalProperties': False, u'type': u'object', u'p

class flexget.plugins.filter.torrent_alive.**TorrentAliveThread**(*tracker*, *info_hash*)
Bases: threading.Thread

run()

flexget.plugins.filter.torrent_alive.**get_http_seeds**(*url*, *info_hash*)

flexget.plugins.filter.torrent_alive.**get_scrape_url**(*tracker_url*, *info_hash*)

flexget.plugins.filter.torrent_alive.**get_tracker_seeds**(*url*, *info_hash*)

flexget.plugins.filter.torrent_alive.**get_udp_seeds**(*url*, *info_hash*)

flexget.plugins.filter.torrent_alive.**max_seeds_from_threads**(*threads*)
Joins the threads and returns the maximum seeds found from any of them.

Parameters **threads** – A list of started *TorrentAliveThread*

Returns Maximum seeds found from any of the threads

flexget.plugins.filter.torrent_alive.**register_plugin**()

generic Package

generic Package

Plugins handling multiple task phases.

archive Module

class flexget.plugins.generic.archive.**Archive**
Bases: object

Archives all new items into database where they can be later searched and injected. Stores the entries in the state as they are at the exit phase, this way task cleanup for title etc is stored into the database. This may however make injecting them back to the original task work wrongly.

on_task_abort(*task*, *config*)
Archive even on task abort, except if the abort has happened before session was started.

on_task_learn(*task*, *config*)
Add new entries into archive. We use learn phase in case the task corrects title or url via some plugins.

schema = {u'oneOf': [{u'type': u'boolean'}, {u'items': {u'type': u'string'}, u'type': u'array'}]}

class flexget.plugins.generic.archive.**ArchiveEntry**
Bases: sqlalchemy.ext.declarative.api.Base

added

description

id

sources

tags

task

title

url

class flexget.plugins.generic.archive.**ArchiveSource**(*name*)
Bases: sqlalchemy.ext.declarative.api.Base

id

name

class flexget.plugins.generic.archive.**ArchiveTag**(*name*)
Bases: sqlalchemy.ext.declarative.api.Base

id

name

class flexget.plugins.generic.archive.**UrlrewriteArchive**
Bases: object

Provides capability to rewrite urls from archive or make searches with discover.

entry_map = {u'url': u'url', u'description': u'description', u'title': u'title'}

schema = {u'oneOf': [{u'type': u'boolean'}, {u'items': {u'type': u'string'}, u'type': u'array'}]}

search(*task*, *entry*, *config=None*)
Search plugin API method

flexget.plugins.generic.archive.**cli_inject**(*manager*, *options*)

flexget.plugins.generic.archive.**cli_search**(*options*)

flexget.plugins.generic.archive.**consolidate**()
Converts previous archive data model to new one.

flexget.plugins.generic.archive.**do_cli**(*manager*, *options*)

flexget.plugins.generic.archive.**get_source**(*name*, *session*)

Parameters

- **name** (*string*) – Source name
- **session** – SQLAlchemy session

Returns ArchiveSource from db or new one

flexget.plugins.generic.archive.**get_tag**(*name*, *session*)

Parameters

- **name** (*string*) – Tag name
- **session** – SQLAlchemy session

Returns ArchiveTag from db or new one

flexget.plugins.generic.archive.**register_parser_arguments**()

flexget.plugins.generic.archive.**register_plugin**()

flexget.plugins.generic.archive.**search**(*session*, *text*, *tags=None*, *sources=None*, *desc=False*)
Search from the archive.

Parameters

- **text** (*string*) – Search text, spaces and dots are tried to be ignored.
- **session** (Session) – SQLAlchemy session, should not be closed while iterating results.
- **tags** (*list*) – Optional list of acceptable tags

- **sources** (*list*) – Optional list of acceptable sources
- **desc** (*bool*) – Sort results descending

Returns ArchiveEntries responding to query

flexget.plugins.generic.archive.**tag_source**(*source_name*, *tag_names=None*)
Tags all archived entries within a source with supplied tags

Parameters

- **source_name** (*string*) – Source name
- **tag_names** (*list*) – List of tag names to add

cron_env Module

flexget.plugins.generic.cron_env.**check_env**(*manager*, *options*)

urlfix Module

class flexget.plugins.generic.urlfix.**UrlFix**
Bases: object

Automatically fix broken urls.

on_task_input(*task*, *config*)

schema = {u'type': u'boolean'}

flexget.plugins.generic.urlfix.**register_plugin**()

welcome Module

flexget.plugins.generic.welcome.**welcome_message**(*manager*)

input Package

input Package

Plugins for "input" task phase.

apple_trailers Module

class flexget.plugins.input.apple_trailers.**AppleTrailers**
Bases: *flexget.plugins.input.rss.InputRSS*

Adds support for Apple.com movie trailers.

Configuration: quality: Set the desired resolution - 480p or 720p. default '720p' genres: List of genres used to filter the entries. If set, the trailer must match at least one listed genre to be accepted. Genres that can be used: Action and Adventure, Comedy, Documentary, Drama, Family, Fantasy, Foreign, Horror, Musical, Romance, Science Fiction, Thriller. default '' (all)

apple_trailers: quality: 720p genres: ['Action and Adventure']

Alternatively, a simpler configuration format can be used. This uses the default genre filter, all:

apple_trailers: 720p

This plugin adds the following fields to the entry: movie_name, movie_year, genres, apple_trailers_name, movie_studio

movie_name: Name of the movie movie_year: Year the movie was/will be released genres: Comma-separated list of genres that apply to the movie apple_trailers_name: Contains the Apple-supplied name of the clip, such as 'Clip 2', 'Trailer', 'Winter Olympic Preview' movie_studio: Name of the studio that makes the movie

on_task_input(**args*, ***kwargs*)

on_task_start(*task*, *config*)

qualities = [u'480p', u'720p']

rss_url = u'http://trailers.apple.com/trailers/home/rss/newtrailers.rss'

schema = {u'oneOf': [{u'additionalProperties': False, u'type': u'object', u'properties': {u'genres': {u'items': {u'type': u

flexget.plugins.input.apple_trailers.**register_plugin**()

backlog Module

class flexget.plugins.input.backlog.**BacklogEntry**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

entry

expire

id

task

title

class flexget.plugins.input.backlog.**InputBacklog**
Bases: object

Keeps task history for given amount of time.

Example:

```
backlog: 4 days
```

Rarely useful for end users, mainly used by other plugins.

add_backlog(**args*, ***kwargs*)

get_injections(**args*, ***kwargs*)

learn_backlog(*task*, *amount=u''*)
Learn current entries into backlog. All task inputs must have been executed.

on_task_abort(*task*, *config*)
Remember all entries until next execution when task gets aborted.

on_task_input(*task*, *config*)

schema = {u'type': u'string', u'format': u'interval'}

flexget.plugins.input.backlog.**register_plugin**()

discover Module

class flexget.plugins.input.discover.**Discover**
Bases: object

Discover content based on other inputs material.

Example:

```
discover:
  what:
    - emit_series: yes
  from:
    - piratebay
  interval: [1 hours|days|weeks]
  ignore_estimations: [yes|no]
```

entry_complete(*entry*, *query=None*, *search_results=None*, ***kwargs*)

estimated(*entries*)

Returns Entries that we have estimated to be available

execute_inputs(*config*, *task*)

Parameters

- **config** – Discover config
- **task** – Current task

Returns List of pseudo entries created by inputs under *what* configuration

execute_searches(*config*, *entries*, *task*)

Parameters

- **config** – Discover plugin config
- **entries** – List of pseudo entries to search
- **task** – Task being run

Returns List of entries found from search engines listed under *from* configuration

interval_expired(*config*, *task*, *entries*)
Maintain some limit levels so that we don't hammer search sites with unreasonable amount of queries.

Returns Entries that are up for `config['interval']`

interval_total_seconds(*interval*)
Because python 2.6 doesn't have total_seconds()

on_task_input(*task*, *config*)

schema = {u'additionalProperties': False, u'required': [u'what', u'from'], u'type': u'object', u'properties': {u'limit': {u

class flexget.plugins.input.discover.**DiscoverEntry**(*title*, *task*)
Bases: sqlalchemy.ext.declarative.api.Base

id

last_execution

task

title

flexget.plugins.input.discover.**db_cleanup**(*manager*, *session*)

flexget.plugins.input.discover.**register_parser_arguments**()

flexget.plugins.input.discover.**register_plugin**()

emit_movie_queue Module

class flexget.plugins.input.emit_movie_queue.**EmitMovieQueue**
Bases: object

Use your movie queue as an input by emitting the content of it

on_task_input(*task*, *config*)

prepare_config(*config*)

schema = {u'oneOf': [{u'type': u'boolean'}, {u'additionalProperties': False, u'type': u'object', u'properties': {u'quality

flexget.plugins.input.emit_movie_queue.**register_plugin**()

emit_series Module

class flexget.plugins.input.emit_series.**EmitSeries**
Bases: object

Emit next episode number from all series configured in this task.

Supports only 'ep' and 'sequence' mode series.

ep_identifiers(*season*, *episode*)

on_search_complete(*entry*, *task=None*, *identified_by=None*, ***kwargs*)
Decides whether we should look for next ep/season based on whether we found/accepted any episodes.

on_task_input(*task*, *config*)

schema = {u'oneOf': [{u'type': u'boolean'}, {u'additionalProperties': False, u'type': u'object', u'properties': {u'backfil

search_entry(*series*, *season*, *episode*, *task*, *rerun=True*)

sequence_identifiers(*episode*)

flexget.plugins.input.emit_series.**register_plugin**()

find Module

class flexget.plugins.input.find.**InputFind**
Bases: object

Uses local path content as an input, recurses through directories and creates entries for files that match mask.

You can specify either the mask key, in shell file matching format, (see python fnmatch module,) or regexp key.

Example:

```
find:
  path: /storage/movies/
  mask: *.avi
```

Example:

```
    find:
      path:
        - /storage/movies/
        - /storage/tv/
      regexp: .*\.(avi|mkv)$
```

on_task_input(*task*, *config*)

prepare_config(*config*)

schema = {u'additionalProperties': False, u'required': [u'path'], u'type': u'object', u'properties': {u'path': {u'oneOf':

flexget.plugins.input.find.**register_plugin**()

gen_series Module

class flexget.plugins.input.gen_series.**GenSeries**
Bases: object

Purely for debugging purposes. Not great quality :)

gen_series_data: series: NUM seasons: NUM episodes: NUM qualities:

- LIST

This will also auto configure series plugin for testing

on_task_exit(*task*, *config*)

on_task_input(*task*, *config*)

on_task_start(*task*, *config*)

schema = {u'type': u'object', u'minProperties': 1}

flexget.plugins.input.gen_series.**register_plugin**()

generate Module

class flexget.plugins.input.generate.**Generate**
Bases: object

Generates n number of random entries. Used for debugging purposes.

on_task_input(*task*, *config*)

schema = {u'type': u'integer'}

flexget.plugins.input.generate.**register_plugin**()

html Module

class flexget.plugins.input.html.**InputHtml**
Bases: object

Parses urls from html page. Usefull on sites which have direct download links of any type (mp3, jpg, torrent, ...).

Many anime-fansubbers do not provide RSS-feed, this works well in many cases.

Configuration expects url parameter.

Note: This returns ALL links on url so you need to configure filters to match only to desired content.

build_config(*config*)

create_entries(*page_url*, *soup*, *config*)

on_task_input(**args*, ***kwargs*)

validator()

flexget.plugins.input.html.**register_plugin**()

imdb_list Module

class flexget.plugins.input.imdb_list.**ImdbList**
Bases: object

"Creates an entry for each movie in your imdb list.

on_task_input(**args*, ***kwargs*)

schema = {u'error_anyOf': u'user_id is required if not using a custom list (lsXXXXXXXXX format)', u'type': u'object',

flexget.plugins.input.imdb_list.**register_plugin**()

inject Module

input_csv Module

class flexget.plugins.input.input_csv.**InputCSV**
Bases: object

Adds support for CSV format. Configuration may seem a bit complex, but this has advantage of being universal solution regardless of CSV and internal entry fields.

Configuration format:

csv: url: <url> values:

<field>: <number>

Example DB-fansubs:

csv: url: http://www.dattebayo.com/t/dump values:

title: 3 # title is in 3th field url: 1 # download url is in 1st field

Fields title and url are mandatory. First field is 1. List of other common (optional) fields can be found from wiki.

on_task_input(**args*, ***kwargs*)

schema = {u'additionalProperties': False, u'required': [u'url', u'values'], u'type': u'object', u'properties': {u'url': {u'ty

flexget.plugins.input.input_csv.**register_plugin**()

inputs Module

class flexget.plugins.input.inputs.**PluginInputs**
Bases: object

Allows the same input plugin to be configured multiple times in a task.

Example:

```
inputs:
  - rss: http://feeda.com
  - rss: http://feedb.com
```

on_task_input(*task*, *config*)

schema = {u'items': {u'allOf': [{u'$ref': u'/schema/plugins?phase=input'}, {u'maxProperties': 1, u'error_maxPropertie

flexget.plugins.input.inputs.**register_plugin**()

listdir Module

Plugin for filesystem tasks.

class flexget.plugins.input.listdir.**Listdir**
Bases: object

Uses local path content as an input.

Example:

```
listdir: /storage/movies/
```

on_task_input(*task*, *config*)

schema = {u'oneOf': [{u'minItems': 1, u'items': {u'title': u'single value', u'type': u'string', u'format': u'path'}, u'type'

flexget.plugins.input.listdir.**register_plugin**()

mock Module

Plugin for mocking task data.

class flexget.plugins.input.mock.**Mock**
Bases: object

Allows adding mock input entries.

Example:

```
mock:
  - {title: foobar, url: http://some.com }
  - {title: mock, url: http://another.com }
```

If url is not given a random url pointing to localhost will be generated.

on_task_input(*task*, *config*)

schema = {u'items': {u'required': [u'title'], u'type': u'object', u'properties': {u'url': {u'type': u'string'}, u'title': {u'typ

flexget.plugins.input.mock.**register_plugin**()

rlslog Module

class flexget.plugins.input.rlslog.**RlsLog**
Bases: object

Adds support for rlslog.net as a feed.

on_task_input(**args*, ***kwargs*)

parse_rlslog(*rlslog_url*, *task*)

Parameters

- **rlslog_url** – Url to parse from
- **task** – Task instance

Returns List of release dictionaries

schema = {u'type': u'string', u'format': u'url'}

flexget.plugins.input.rlslog.**register_plugin**()

rss Module

class flexget.plugins.input.rss.**InputRSS**

Bases: object

Parses RSS feed.

Hazzlefree configuration for public rss feeds:

```
rss: <url>
```

Configuration with basic http authentication:

```
rss:
  url: <url>
  username: <name>
  password: <password>
```

Advanced usages:

You may wish to clean up the entry by stripping out all non-ascii characters. This can be done by setting ascii value to yes.

Example:

```
rss:
  url: <url>
  ascii: yes
```

In case RSS-feed uses some nonstandard field for urls and automatic detection fails you can configure plugin to use url from any feedparser entry attribute.

Example:

```
rss:
  url: <url>
  link: guid
```

If you want to keep information in another rss field attached to the flexget entry, you can use the other_fields option.

Example:

```
rss:
  url: <url>
  other_fields: [date]
```

You can disable few possibly annoying warnings by setting silent value to yes on feeds where there are frequently invalid items.

Example:

```
rss:
  url: <url>
  silent: yes
```

You can group all the links of an item, to make the download plugin tolerant to broken urls: it will try to download each url until one works. Links are enclosures plus item fields given by the link value, in that order. The value to set is "group_links".

Example:

```
rss:
  url: <url>
  group_links: yes
```

add_enclosure_info(*entry*, *enclosure*, *filename=True*, *multiple=False*)
Stores information from an rss enclosure into an Entry.

build_config(*config*)
Set default values to config

on_task_input(**args*, ***kwargs*)

process_invalid_content(*task*, *data*, *url*)
If feedparser reports error, save the received data and log error.

schema = {u'additionalProperties': False, u'anyOf': [{u'format': u'url'}, {u'format': u'file'}], u'required': [u'url'], u'ty

flexget.plugins.input.rss.**fp_field_name**(*name*)
Translates literal field name to the sanitized one feedparser will use.

flexget.plugins.input.rss.**register_plugin**()

scenereleases Module

tail Module

class flexget.plugins.input.tail.**InputTail**
Bases: object

Parse any text for entries using regular expression.

```
file: <file>
entry:
  <field>: <regexp to match value>
format:
  <field>: <python string formatting>
```

Note: each entry must have at least two fields, title and url

You may wish to specify encoding used by file so file can be properly decoded. List of encodings at http://docs.python.org/library/codecs.html#standard-encodings.

Example:

```
tail:
  file: ~/irclogs/some/log
  entry:
    title: 'TITLE: (.*) URL:'
    url: 'URL: (.*)'
  encoding: utf8
```

format_entry(*entry*, *d*)

on_task_input(*task*, *config*)

schema = {u'additionalProperties': False, u'required': [u'file', u'entry'], u'type': u'object', u'properties': {u'entry': {u'

class flexget.plugins.input.tail.**TailPosition**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

filename

id

position

task

flexget.plugins.input.tail.**register_parser_arguments**()

flexget.plugins.input.tail.**register_plugin**()

text Module

Plugin for text file or URL feeds via regex.

class flexget.plugins.input.text.**Text**
Bases: object

Parse any text for entries using regular expression.

Example:

```
url: <url>
entry:
  <field>: <regexp to match value>
format:
  <field>: <python string formatting>
```

Note: each entry must have atleast two fields, title and url

Example:

```
text:
  url: http://www.nbc.com/Heroes/js/novels.js
  entry:
    title: novelTitle = "(.*)"
    url: novelPrint = "(.*)"
  format:
    url: http://www.nbc.com%(url)s
```

format_entry(*entry*, *d*)

on_task_input(**args*, ***kwargs*)

schema = {u'required': [u'entry', u'url'], u'type': u'object', u'properties': {u'url': {u'oneOf': [{u'type': u'string', u'for

flexget.plugins.input.text.**register_plugin**()

thetvdb_favorites Module

trakt_list Module

class flexget.plugins.input.trakt_list.**TraktList**
Bases: object

Creates an entry for each item in your trakt list.

Syntax:

trakt_list: username: <value> password: <value> type: <shows|movies|episodes> list: <collection|watchlist|watched|custom list name> strip_dates: <yes|no>

Options username, type and list are required. password is required for private lists.

on_task_input(**args*, ***kwargs*)

schema = {u'required': [u'username', u'type', u'list'], u'error_not': u'‘collection‘ and ‘watched‘ lists do not support ‘ep

flexget.plugins.input.trakt_list.**register_plugin**()

tvtorrents Module

local Package

local Package

Plugin package to place local private extensions in.

Use this to avoid naming conflicts with standard plugins and place your own private plugins in "~/.flexget/plugins/local". It's also a good idea to start their names with "**local_**", i.e. use code like this:

```
from flexget import plugin

class LocalPlugin(plugin.Plugin):
    ...
```

metainfo Package

metainfo Package

Plugins for "metainfo" task phase.

content_size Module

class flexget.plugins.metainfo.content_size.**MetainfoContentSize**
Bases: object

Utility:

Check if content size is mentioned in description and set content_size attribute for entries if it is. Also sets content_size for entries with local files from input_listdir.

on_task_metainfo(*task*, *config*)

schema = {u'default': False, u'type': u'boolean'}

flexget.plugins.metainfo.content_size.**register_plugin**()

task Module

class flexget.plugins.metainfo.task.**MetainfoTask**
Bases: object

Set 'task' field for entries.

on_task_metainfo(*task*, *config*)

schema = {u'type': u'boolean'}

flexget.plugins.metainfo.task.**register_plugin**()

imdb_lookup Module

class flexget.plugins.metainfo.imdb_lookup.**Actor**(*imdb_id*, *name=None*)
Bases: sqlalchemy.ext.declarative.api.Base

id

imdb_id

name

class flexget.plugins.metainfo.imdb_lookup.**Director**(*imdb_id*, *name=None*)
Bases: sqlalchemy.ext.declarative.api.Base

id

imdb_id

name

class flexget.plugins.metainfo.imdb_lookup.**Genre**(*name*)
Bases: sqlalchemy.ext.declarative.api.Base

id

name

class flexget.plugins.metainfo.imdb_lookup.**ImdbLookup**
Bases: object

Retrieves imdb information for entries.

Example:

imdb_lookup: yes

Also provides imdb lookup functionality to all other imdb related plugins.

field_map = {u'imdb_genres': <function <lambda> at 0x7f56bced7050>, u'imdb_photo': u'photo', u'movie_name': u't

imdb_id_lookup(**args*, ***kwargs*)

lazy_loader(*entry*)
Does the lookup for this entry and populates the entry fields.

lookup(**args*, ***kwargs*)

on_task_metainfo(*task*, *config*)

register_lazy_fields(*entry*)

schema = {u'type': u'boolean'}

class flexget.plugins.metainfo.imdb_lookup.**Language**(*name*)
Bases: sqlalchemy.ext.declarative.api.Base

id

name

class flexget.plugins.metainfo.imdb_lookup.**Movie**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

actors

directors

expired

Returns True if movie details are considered to be expired, ie. need of update

genres

id

imdb_id

languages

mpaa_rating

original_title

photo

plot_outline

score

title

updated

url

votes

year

class flexget.plugins.metainfo.imdb_lookup.**MovieLanguage**(*language*, *prominence=None*)
Bases: sqlalchemy.ext.declarative.api.Base

language

language_id

movie_id

prominence

class flexget.plugins.metainfo.imdb_lookup.**SearchResult**(*title*, *url=None*)
Bases: sqlalchemy.ext.declarative.api.Base

fails

id

imdb_id

queried

title

url

flexget.plugins.metainfo.imdb_lookup.**register_plugin**()

imdb_url Module

class flexget.plugins.metainfo.imdb_url.**MetainfoImdbUrl**
Bases: object

Scan entry information for imdb url.

on_task_metainfo(*task*, *config*)

schema = {u'type': u'boolean'}

flexget.plugins.metainfo.imdb_url.**register_plugin**()

nzb_size Module

class flexget.plugins.metainfo.nzb_size.**NzbSize**
Bases: object

Provides entry size information when dealing with nzb files

on_task_modify(*task*, *config*)
The downloaded file is accessible in modify phase

flexget.plugins.metainfo.nzb_size.**register_plugin**()

quality Module

class flexget.plugins.metainfo.quality.**MetainfoQuality**
Bases: object

Utility:

Set quality attribute for entries.

get_quality(*entry*)

on_task_metainfo(*task*, *config*)

schema = {u'type': u'boolean'}

flexget.plugins.metainfo.quality.**register_plugin**()

series Module

class flexget.plugins.metainfo.series.**MetainfoSeries**
Bases: object

Check if entry appears to be a series, and populate series info if so.

guess_entry(*entry*, *allow_seasonless=False*, *config=None*)
Populates series_* fields for entries that are successfully parsed.

Parameters **config** (*dict*) – A series config to be used. This will also cause 'path' and 'set' fields to be populated.

on_task_metainfo(*task*, *config*)

schema = {u'type': u'boolean'}

flexget.plugins.metainfo.series.**register_plugin**()

thetvdb_lookup Module

tmdb_lookup Module

class flexget.plugins.metainfo.tmdb_lookup.**PluginTmdbLookup**
Bases: object

Retrieves tmdb information for entries.

Example: tmdb_lookup: yes

field_map = {u'tmdb_tagline': u'tagline', u'tmdb_budget': u'budget', u'tmdb_trailer': u'trailer', u'tmdb_popularity':

lazy_loader(*entry*)
Does the lookup for this entry and populates the entry fields.

lookup(*entry*)
Populates all lazy fields to an Entry. May be called by other plugins requiring tmdb info on an Entry

Parameters **entry** – Entry instance

on_task_metainfo(*task*, *config*)

schema = {u'type': u'boolean'}

flexget.plugins.metainfo.tmdb_lookup.**register_plugin**()

torrent_size Module

class flexget.plugins.metainfo.torrent_size.**TorrentSize**
Bases: object

Provides file size information when dealing with torrents

on_task_modify(*task*, *config*)

flexget.plugins.metainfo.torrent_size.**register_plugin**()

modify Package

modify Package

Modification phase plugins.

extension Module

class flexget.plugins.modify.extension.**ModifyExtension**
Bases: object

Allows specifying file extension explicitly when all other built-in detection mechanisms fail.

Example:

extension: nzb

on_task_modify(*task*, *config*)

schema = {u'type': [u'string', u'number']}

flexget.plugins.modify.extension.**register_plugin**()

headers Module

class flexget.plugins.modify.headers.**HTTPHeadersProcessor**(*headers=None*)
Bases: urllib2.BaseHandler

handler_order = 490

http_request(*request*)

http_response(*request*, *response*)

https_request(*request*)

https_response(*request*, *response*)

class flexget.plugins.modify.headers.**PluginHeaders**
Bases: object

Allow setting up any headers in all requests (which use urllib2)

Example:

headers: cookie: uid=<YOUR UID>; pass=<YOUR PASS>

on_task_abort(*task*, *config*)
Task exiting, remove additions

on_task_exit(*task*, *config*)
Task exiting, remove additions

on_task_start(*task*, *config*)
Task starting

schema = {u'additionalProperties': {u'type': u'string'}, u'type': u'object'}

flexget.plugins.modify.headers.**register_plugin**()

manipulate Module

class flexget.plugins.modify.manipulate.**Manipulate**
Bases: object

Usage:

manipulate:

- **<destination field>:** [phase]: <phase> [from]: <source field> [extract]: <regexp> [separator]: <text> [replace]:

  regexp: <regexp> format: <regexp>

  [remove]: <boolean>

Example:

manipulate:

- **title:** extract: [dddd](.*)

on_task_filter(*task*, *config*)

on_task_metainfo(*task*, *config*)

on_task_start(*task*, *config*)
Separates the config into a dict with a list of jobs per phase. Allows us to skip phases without any jobs in them.

process(*entry*, *jobs*)
Process given jobs from config for an entry.

Parameters

- **entry** – Entry to modify
- **jobs** – Config items to run on this entry

Returns True if any fields were modified

validator()

flexget.plugins.modify.manipulate.**register_plugin**()

path_by_ext Module

class flexget.plugins.modify.path_by_ext.**PluginPathByExt**
Bases: object

Allows specifying path based on content-type

Example:

path_by_ext: torrent: ~/watch/torrent/ nzb: ~/watch/nzb/

ext(*task*, *config*, *callback*)

on_task_modify(*task*, *config*)

schema = {u'type': u'object'}

set_path(*entry*, *path*)

flexget.plugins.modify.path_by_ext.**register_plugin**()

plugin_priority Module

class flexget.plugins.modify.plugin_priority.**PluginPriority**
Bases: object

Allows modifying plugin priorities from default values.

Example:

plugin_priority: ignore: 50 series: 100

on_task_abort(*task*, *config*)

on_task_exit(*task*, *config*)

on_task_start(*task*, *config*)

schema = {u'additionalProperties': {u'type': u'integer'}, u'type': u'object'}

flexget.plugins.modify.plugin_priority.**register_plugin**()

set_field Module

class flexget.plugins.modify.set_field.**ModifySet**
Bases: object

Allows adding information to a task entry for use later.

Example:

set: path: ~/download/path/

lazy_set(*config*, *field*, *orig_field_value*, *entry*, *errors=True*)

modify(*entry*, *config*, *errors=True*)
This can be called from a plugin to add set values to an entry

on_task_metainfo(*task*, *config*)
Adds the set dict to all accepted entries.

schema = {u'type': u'object', u'minProperties': 1}

flexget.plugins.modify.set_field.**register_plugin**()

torrent Module

class flexget.plugins.modify.torrent.**TorrentFilename**
Bases: object

Makes sure that entries containing torrent-file have .torrent extension. This is enabled always by default (builtins).

TORRENT_PRIO = 255

make_filename(*torrent*, *entry*)
Build a filename for this torrent

on_task_modify(*task*, *config*)

on_task_output(*task*, *config*)

purge(*entry*)

flexget.plugins.modify.torrent.**register_plugin**()

torrent_scrub Module

Torrent Scrubber Plugin.

class flexget.plugins.modify.torrent_scrub.**TorrentScrub**
Bases: object

Scrubs torrents from unwanted keys.

Example:

tasks:

rutorrent-fast-resume-infected-task: torrent_scrub: resume

RT_KEYS = (u'libtorrent_resume', u'log_callback', u'err_callback', u'rtorrent')

SCRUB_MODES = (u'off', u'on', u'all', u'resume', u'rtorrent')

SCRUB_PRIO = 245

on_task_modify(*task*, *config*)
Scrub items that are torrents, if they're affected.

schema = {u'oneOf': [{u'type': u'boolean'}, {u'enum': [u'off', u'on', u'all', u'resume', u'rtorrent'], u'type': u'string'}, {

flexget.plugins.modify.torrent_scrub.**register_plugin**()

trackers Module

class flexget.plugins.modify.trackers.**AddTrackers**
Bases: object

Adds tracker URL to torrent files.

Configuration example:

add_trackers:

- uri://tracker_address:port/

This will add all tracker URL uri://tracker_address:port/.

on_task_modify(*task*, *config*)

schema = {u'items': {u'type': u'string', u'format': u'url'}, u'type': u'array'}

class flexget.plugins.modify.trackers.**ModifyTrackers**
Bases: object

Modify tracker URL to torrent files.

Configuration example:

```
modify_trackers:
  - SearchAndReplace:
      from: string_to_search
      to: string_to_replace
```

on_task_modify(*task*, *config*)

schema = {u'items': {u'additionalProperties': {u'additionalProperties': False, u'type': u'object', u'properties': {u'to': {

class flexget.plugins.modify.trackers.**RemoveTrackers**
Bases: object

Removes trackers from torrent files using regexp matching.

Configuration example:

remove_trackers:

- moviex

This will remove all trackers that contain text moviex in their url.

on_task_modify(*task*, *config*)

schema = {u'items': {u'type': u'string', u'format': u'regex'}, u'type': u'array'}

flexget.plugins.modify.trackers.**register_plugin**()

operate Package

operate Package

disable_builtins Module

disable_phases Module

class flexget.plugins.operate.disable_phases.**PluginDisablePhases**
Bases: object

Disables phases from task execution.

Mainly meant for advanced users and development.

Example:

disable_phases:

- download

on_task_start(*task*, *config*)

schema

flexget.plugins.operate.disable_phases.**register_plugin**()

task_execution Module

task_priority Module

free_space Module

class flexget.plugins.operate.free_space.**PluginFreeSpace**
Bases: object

Aborts a task if an entry is accepted and there is less than a certain amount of space free on a drive.

on_task_download(*task*, *config*)

prepare_config(*config*)

schema = {u'oneOf': [{u'type': u'number'}, {u'additionalProperties': False, u'required': [u'space'], u'type': u'object', u

flexget.plugins.operate.free_space.**get_free_space**(*folder*)
Return folder/drive free space (in megabytes)

flexget.plugins.operate.free_space.**register_plugin**()

interval Module

class flexget.plugins.operate.interval.**PluginInterval**
Bases: object

Allows specifying minimum interval for task execution.

Format: [n] [minutes|hours|days|weeks]

Example:

interval: 7 days

on_task_start(*task*, *config*)

schema = {u'type': u'string', u'format': u'interval'}

flexget.plugins.operate.interval.**register_parser_arguments**()

flexget.plugins.operate.interval.**register_plugin**()

max_reruns Module

class flexget.plugins.operate.max_reruns.**MaxReRuns**
Bases: object

Overrides the maximum amount of re-runs allowed by a task.

on_task_abort(*task*, *config*)

on_task_exit(*task*, *config*)

on_task_start(*task*, *config*)

schema = {u'type': u'integer'}

flexget.plugins.operate.max_reruns.**register_plugin**()

sequence Module

class flexget.plugins.operate.sequence.**PluginSequence**
Bases: object

Allows the same plugin to be configured multiple times in a task.

Example: sequence:

- rss: http://feeda.com
- rss: http://feedb.com

schema = {u'items': {u'$ref': u'/schema/plugins'}, u'type': u'array'}

flexget.plugins.operate.sequence.**register_plugin**()

sleep Module

class flexget.plugins.operate.sleep.**PluginSleep**
Bases: object

Causes a pause to occur during the specified phase of a task

do_sleep(*config*, *phase*)

on_task_abort(*task*, *config*)

on_task_download(*task*, *config*)

on_task_exit(*task*, *config*)

on_task_filter(*task*, *config*)

on_task_input(*task*, *config*)

on_task_learn(*task*, *config*)

on_task_metainfo(*task*, *config*)

on_task_modify(*task*, *config*)

on_task_output(*task*, *config*)

on_task_start(*task*, *config*)

schema = {u'oneOf': [{u'additionalProperties': False, u'required': [u'seconds'], u'type': u'object', u'properties': {u'sec

flexget.plugins.operate.sleep.**register_plugin**()

output Package

output Package

Plugins for "output" task phase.

download Module

class flexget.plugins.output.download.**PluginDownload**
Bases: object

Downloads content from entry url and writes it into a file.

Example:

```
download: ~/torrents/
```

Allow HTML content:

By default download plugin reports failure if received content is a html. Usually this is some sort of custom error page without proper http code and thus entry is assumed to be downloaded incorrectly.

In the rare case you actually need to retrieve html-pages you must disable this feature.

Example:

```
download:
  path: ~/something/
  fail_html: no
```

You may use commandline parameter –dl-path to temporarily override all paths to another location.

cleanup_temp_file(*entry*)

cleanup_temp_files(*task*)
Checks all entries for leftover temp files and deletes them.

download_entry(*task*, *entry*, *url*, *tmp_path*)
Downloads *entry* by using *url*.

Raises Several types of exceptions ...

Raises PluginWarning

filename_ext_from_mime(*entry*)
Tries to set filename extension from mime-type

filename_from_headers(*entry*, *response*)
Checks entry filename if it's found from content-disposition

get_temp_file(*task*, *entry*, *require_path=False*, *handle_magnets=False*, *fail_html=True*, *tmp_path='/tmp'*)
Download entry content and store in temporary folder. Fails entry with a reason if there was problem.

Parameters

- **require_path** (*bool*) – whether or not entries without 'path' field are ignored
- **handle_magnets** (*bool*) – when used any of urls containing magnet link will replace url, otherwise warning is printed.
- **fail_html** – fail entries which url respond with html content
- **tmp_path** – path to use for temporary files while downloading

get_temp_files(*task*, *require_path=False*, *handle_magnets=False*, *fail_html=True*, *tmp_path='/tmp'*)
Download all task content and store in temporary folder.

Parameters

- **require_path** (*bool*) – whether or not entries without 'path' field are ignored
- **handle_magnets** (*bool*) – when used any of urls containing magnet link will replace url, otherwise warning is printed.
- **fail_html** – fail entries which url respond with html content
- **tmp_path** – path to use for temporary files while downloading

on_task_abort(*task*, *config*)
Make sure all temp files are cleaned up when task is aborted.

on_task_download(*task*, *config*)

on_task_learn(*task*, *config*)
Make sure all temp files are cleaned up after output phase

on_task_output(*task*, *config*)
Move downloaded content from temp folder to final destination

output(*task*, *entry*, *config*)
Moves temp-file into final destination

Raises: PluginError if operation fails

process_config(*config*)
Return plugin configuration in advanced form

process_entry(*task*, *entry*, *url*, *tmp_path*)
Processes *entry* by using *url*. Does not use entry['url']. Does not fail the *entry* if there is a network issue, instead just logs and returns a string error.

Parameters

- **task** – Task
- **entry** – Entry
- **url** – Url to try download
- **tmp_path** – Path to store temporary files

Returns String error, if failed.

save_error_page(*entry*, *task*, *page*)

schema = {u'oneOf': [{u'additionalProperties': False, u'type': u'object', u'properties': {u'temp': {u'type': u'string', u'f

flexget.plugins.output.download.**register_parser_arguments**()

flexget.plugins.output.download.**register_plugin**()

dump Module

class flexget.plugins.output.dump.**OutputDump**
Bases: object

Outputs all entries to console

on_task_output(*task*, *config*)

schema = {u'type': u'boolean'}

flexget.plugins.output.dump.**dump**(*entries*, *debug=False*, *eval_lazy=False*, *trace=False*, *title_only=False*)
Dump *entries* to stdout

Parameters

- **entries** (*list*) – Entries to be dumped.
- **debug** (*bool*) – Print non printable fields as well.
- **eval_lazy** (*bool*) – Evaluate lazy fields.
- **trace** (*bool*) – Display trace information.
- **title_only** (*bool*) – Display only title field

flexget.plugins.output.dump.**register_parser_arguments**()

flexget.plugins.output.dump.**register_plugin**()

dump_config Module

class flexget.plugins.output.dump_config.**OutputDumpConfig**
Bases: object

Dumps task config in STDOUT in yaml at exit or abort event.

on_task_start(*task*, *config*)

flexget.plugins.output.dump_config.**register_parser_arguments**()

flexget.plugins.output.dump_config.**register_plugin**()

exec Module

class flexget.plugins.output.exec.**EscapingDict**(*mapping*)
Bases: _abcoll.MutableMapping

Helper class, same as a dict, but returns all string value with quotes escaped.

class flexget.plugins.output.exec.**PluginExec**
Bases: object

Execute commands

Simple example, xecute command for entries that reach output:

```
exec: echo 'found {{title}} at {{url}}' > file
```

Advanced Example:

```
exec:
  on_start:
    phase: echo "Started"
  on_input:
    for_entries: echo 'got {{title}}'
  on_output:
    for_accepted: echo 'accepted {{title}} - {{url}} > file
```

You can use all (available) entry fields in the command.

HANDLED_PHASES = [u'start', u'input', u'filter', u'output', u'exit']

NAME = u'exec'

execute(*task*, *phase_name*, *config*)

execute_cmd(*cmd*, *allow_background*, *encoding*)

prepare_config(*config*)

schema = {u'definitions': {u'phaseSettings': {u'additionalProperties': False, u'type': u'object', u'properties': {u'phase':

flexget.plugins.output.exec.**register_plugin**()

history Module

html Module

class flexget.plugins.output.html.**OutputHtml**

on_task_output(*task*, *config*)

schema = {u'additionalProperties': False, u'required': [u'file'], u'type': u'object', u'properties': {u'file': {u'type': u'stri

flexget.plugins.output.html.**register_plugin**()

move Module

class flexget.plugins.output.move.**BaseFileOps**
Bases: object

clean_source(*task*, *config*, *entry*)

log = None

on_task_output(*task*, *config*)

class flexget.plugins.output.move.**CopyFiles**
Bases: *flexget.plugins.output.move.TransformingOps*

Copy all accepted files.

destination_field = u'copy_to'

log = <flexget.logger.FlexGetLogger object>

move = False

schema = {u'oneOf': [{u'type': u'boolean'}, {u'additionalProperties': False, u'type': u'object', u'properties': {u'unpack

class flexget.plugins.output.move.**DeleteFiles**
Bases: *flexget.plugins.output.move.BaseFileOps*

Delete all accepted files.

handle_entry(*task*, *config*, *entry*, *siblings*)

log = <flexget.logger.FlexGetLogger object>

schema = {u'oneOf': [{u'type': u'boolean'}, {u'additionalProperties': False, u'type': u'object', u'properties': {u'clean_s

class flexget.plugins.output.move.**MoveFiles**
Bases: *flexget.plugins.output.move.TransformingOps*

Move all accepted files.

destination_field = u'move_to'

log = <flexget.logger.FlexGetLogger object>

move = True

schema = {u'oneOf': [{u'type': u'boolean'}, {u'additionalProperties': False, u'type': u'object', u'properties': {u'clean_s

class flexget.plugins.output.move.**TransformingOps**
Bases: *flexget.plugins.output.move.BaseFileOps*

destination_field = None

handle_entry(*task*, *config*, *entry*, *siblings*)

move = None

flexget.plugins.output.move.**get_directory_size**(*directory*)

Parameters **directory** – Path

Returns Size in bytes (recursively)

flexget.plugins.output.move.**register_plugin**()

notifymyandroid Module

class flexget.plugins.output.notifymyandroid.**OutputNotifyMyAndroid**
Bases: object

Example:

```
notifymyandroid:
  apikey: xxxxxxx
  [application: application name, default FlexGet]
  [event: event title, default New Release]
  [priority: -2 - 2 (2 = highest), default 0]
```

Configuration parameters are also supported from entries (eg. through set).

on_task_output(*task*, *config*)

schema = {u'additionalProperties': False, u'required': [u'apikey'], u'type': u'object', u'properties': {u'priority': {u'defa

flexget.plugins.output.notifymyandroid.**register_plugin**()

prowl Module

class flexget.plugins.output.prowl.**OutputProwl**
Bases: object

Send prowl notifications

Example:

```
prowl:
  apikey: xxxxxxx
  [application: application name, default FlexGet]
  [event: event title, default New Release]
  [priority: -2 - 2 (2 = highest), default 0]
  [description: notification to send]
```

Configuration parameters are also supported from entries (eg. through set).

on_task_output(*task*, *config*)

schema = {u'additionalProperties': False, u'required': [u'apikey'], u'type': u'object', u'properties': {u'priority': {u'defa

flexget.plugins.output.prowl.**register_plugin**()

pyload Module

class flexget.plugins.output.pyload.**PluginPyLoad**
Bases: object

Parse task content or url for hoster links and adds them to pyLoad.

Example:

```
pyload:
  api: http://localhost:8000/api
  queue: yes
  username: my_username
  password: my_password
  folder: desired_folder
```

```
    package: desired_package_name (jinja2 supported)
    hoster:
      - YoutubeCom
    parse_url: no
    multiple_hoster: yes
    enabled: yes
```

Default values for the config elements:

```
pyload:
    api: http://localhost:8000/api
    queue: no
    hoster: ALL
    parse_url: no
    multiple_hoster: yes
    enabled: yes
```

DEFAULT_API = u'http://localhost:8000/api'

DEFAULT_FOLDER = u''

DEFAULT_HANDLE_NO_URL_AS_FAILURE = False

DEFAULT_HOSTER = []

DEFAULT_MULTIPLE_HOSTER = True

DEFAULT_PARSE_URL = False

DEFAULT_PREFERRED_HOSTER_ONLY = False

DEFAULT_QUEUE = False

add_entries(*task*, *config*)
Adds accepted entries

on_task_output(*task*, *config*)

schema = {u'oneOf': [{u'type': u'boolean'}, {u'additionalProperties': False, u'type': u'object', u'properties': {u'usernar

class flexget.plugins.output.pyload.**PyloadApi**(*requests*, *url*)
Bases: object

get_session(*config*)

query(*method*, *post=None*)

flexget.plugins.output.pyload.**register_plugin**()

queue_movies Module

rss Module

class flexget.plugins.output.rss.**OutputRSS**
Bases: object

Write RSS containing succeeded (downloaded) entries.

Example:

```
make_rss: ~/public_html/flexget.rss
```

You may write into same file in multiple tasks.

Example:

```
my-task-A:
  make_rss: ~/public_html/series.rss
  .
  .
my-task-B:
  make_rss: ~/public_html/series.rss
  .
  .
```

With this example file series.rss would contain succeeded entries from both tasks.

Number of days / items

By default output contains items from last 7 days. You can specify different perioid, number of items or both. Value -1 means unlimited.

Example:

```
make_rss:
  file: ~/public_html/series.rss
  days: 2
  items: 10
```

Generate RSS that will containing last two days and no more than 10 items.

Example 2:

```
make_rss:
  file: ~/public_html/series.rss
  days: -1
  items: 50
```

Generate RSS that will contain last 50 items, regardless of dates.

RSS location link:

You can specify the url location of the rss file.

Example:

```
make_rss:
  file: ~/public_html/series.rss
  rsslink: http://my.server.net/series.rss
```

RSS link

You can specify what field from entry is used as a link in generated rss feed.

Example:

```
make_rss:
  file: ~/public_html/series.rss
  link:
    - imdb_url
```

List should contain a list of fields in order of preference. Note that the url field is always used as last possible fallback even without explicitly adding it into the list.

Default list: imdb_url, input_url, url

on_task_exit(*task*, *config*)
Store finished / downloaded entries at exit

on_task_output(*task*, *config*)

prepare_config(*config*)

schema = {u'oneOf': [{u'type': u'string'}, {u'additionalProperties': False, u'required': [u'file'], u'type': u'object', u'pro

class flexget.plugins.output.rss.**RSSEntry**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

description

file

id

link

published

rsslink

title

flexget.plugins.output.rss.**register_plugin**()

sabnzbd Module

class flexget.plugins.output.sabnzbd.**OutputSabnzbd**
Bases: object

Example:

```
sabnzbd:
  apikey: 123456
  url: http://localhost/sabnzbd/api?
  category: movies
```

All parameters:

```
sabnzbd:
  apikey: ...
  url: ...
  category: ...
  script: ...
  pp: ...
  priority: ...
```

get_params(*config*)

on_task_output(*task*, *config*)

schema = {u'additionalProperties': False, u'required': [u'key', u'url'], u'type': u'object', u'properties': {u'category': {u

flexget.plugins.output.sabnzbd.**register_plugin**()

send_email Module

class flexget.plugins.output.send_email.**OutputEmail**
Bases: object

Send an e-mail with the list of all succeeded (downloaded) entries.

Configuration options

Option	Description
from	The email address from which the email will be sent (required)
to	The email address of the recipient (required)
smtp_host	The host of the smtp server
smtp_port	The port of the smtp server
smtp_username	The username to use to connect to the smtp server
smtp_password	The password to use to connect to the smtp server
smtp_tls	Should we use TLS to connect to the smtp server
smtp_ssl	Should we use SSL to connect to the smtp server Due to a bug in python, this only works in python 2.6.3 and up
active	Is this plugin active or not

Config basic example:

```
email:
  from: xxx@xxx.xxx
  to: xxx@xxx.xxx
  smtp_host: smtp.host.com
```

Config example with smtp login:

```
email:
  from: xxx@xxx.xxx
  to: xxx@xxx.xxx
  smtp_host: smtp.host.com
  smtp_port: 25
  smtp_login: true
  smtp_username: my_smtp_login
  smtp_password: my_smtp_password
  smtp_tls: true
```

Config multi-task example:

```
global:
  email:
    from: xxx@xxx.xxx
    to: xxx@xxx.xxx
    smtp_host: smtp.host.com

tasks:
  task1:
    rss: http://xxx
  task2:
    rss: http://yyy
    email:
      active: False
  task3:
    rss: http://zzz
    email:
      to: zzz@zzz.zzz
```

GMAIL example:

```
from: from@gmail.com
to: to@gmail.com
smtp_host: smtp.gmail.com
```

```
    smtp_port: 587
    smtp_login: true
    smtp_username: gmailUser
    smtp_password: gmailPassword
    smtp_tls: true
```

Default values for the config elements:

```
    email:
      active: True
      smtp_host: localhost
      smtp_port: 25
      smtp_login: False
      smtp_username:
      smtp_password:
      smtp_tls: False
      smtp_ssl: False
```

on_task_abort(*task*, *config*)

on_task_output(*task*, *config*)

schema = {u'additionalProperties': False, u'required': [u'to', u'from'], u'type': u'object', u'properties': {u'smtp_passw

flexget.plugins.output.send_email.**global_send**(*manager*, *options*)

flexget.plugins.output.send_email.**prepare_config**(*config*)

flexget.plugins.output.send_email.**register_config_key**()

flexget.plugins.output.send_email.**register_plugin**()

flexget.plugins.output.send_email.**send_email**(*subject*, *content*, *config*)
Send email at exit.

flexget.plugins.output.send_email.**setup**(*manager*, *options*)

subtitles Module

class flexget.plugins.output.subtitles.**Subtitles**
Bases: object

Fetch subtitles from opensubtitles.org

on_task_download(*task*, *config*)

prepare_config(*config*, *task*)

schema = {u'additionalProperties': False, u'type': u'object', u'properties': {u'languages': {u'default': [u'eng'], u'items'

flexget.plugins.output.subtitles.**register_plugin**()

services Package

services Package

myepisodes Module

trakt_acquired Module

start Package

start Package

Plugins for "start" task phase.

f

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y