Burst Documentation

Release 0.5.10

elgatito

Dec 13, 2017

Contents

A burst of providers.

CHAPTER 1

Features

- Fast, very fast
- Compatible with Magnetic/Magnetizer, but **disable the Elementum Connector**
- Can extract providers, parsers and settings from Magnetic
- No extra add-ons to install, all providers are included
- No extra service running in the background
- Easy settings to enable or disable providers and filters
- First-class support with Elementum, and only Elementum (don't even ask)
- Simple definitions-based architecture with overrides
- Clean, PEP8 compliant code

CHAPTER 2

Installation

IMPORTANT: Disable the Magnetic Elementum Connector before anything else.

Get the latest release from https://burst.surge.sh and *Install from zip* within Kodi, or install the add-on from the Elementum Repository if you already have Elementum installed.

CHAPTER 3

Topics

3.1 Using filters

If you go in the add-on's Advanced settings, you will notice an option named `Additional keyword filters (comma separated)`. Enabling this option will bring up three sub-settings: `Accept`, `Block` and `Require`. They all expect the same kind of parameters, which is a comma-separated list of keywords to respectively either accept, block or require. Although it's mostly self-explanatory, let's go over each of them to fully understand how they behave, and what kind of results you mind expect when using those settings.

3.1.1 Format

A comma-separated list is a standard way of defining multiple values. You can include spaces between keywords for readability, and Burst will work just the same. For example, those two settings will be equivalent: `HEVC,H265` vs `HEVC, H265`. They will both be understood as a list with the values `["HEVC", "H265"]`. Also note that uppercase or lowercase makes no difference, so both `hevc` and `HeVc` in a result name would also be considered a match.

The only special trick about the format of keywords is done by using underscores (_), which tell Burst to make sure there is a space, dot, dash, also an underscore, or other separator between your keyword and the other parts of the result's name. For example, if you want to match `ITA`, but not `italian`, you would use `_ITA_` as your keyword, which would match names like `A.Movie.2017.ITA.720p` but *not* `A.Movie.2017.Italian.720p`. A trailing underscore would also return a match, ie. `A.Movie.720p.ITA`. **Note that the 'Require' keyword treats underscores literally**, so using `_ITA_` in *Require* would only match names like `A.Movie_ITA_720p`.

3.1.2 Keyword types

Accept

The *Accept* setting will return results that include **any** of the keywords you specify. For example, `Italian, French` will return results that either include `italian` or `french`.

Block

The *Block* setting will block results that include **any** of the keywords you specify, and can be the most dangerous filter to use. For example, `ITA` would block every result that has `ita` anywhere in its name, regardless of delimiters like dots and dashes, so if you're looking for a movie named *My Inheritance*, you would get absolutely no result. For that reason, you should usually always add underscores around *Block* keywords to make sure there are delimiters around those keywords.

Require

The *Require* setting is also a dangerous filter to use, and will require **all** the keywords you specify to be included in the result names. For example, if you specify `ITA, _FR_`, you would only get results that include **both** `ITA` and `FR` (with delimiters), which will be very few if any. It can however be a very useful setting to get only results that include your preferred language.

3.2 Using overrides

Default fixes and overrides are located in `burst/providers/definitions.py`, and although you can edit that file directly, keep in mind that you will lose your changes on the next update. You can override existing definitions by adding another file named `overrides.py` in your userdata folder under `addon_data/script.elementum.burst`. Put all your overrides in the `overrides` variable within that file, as such:

```
overrides = {
    'torlock': {
        'name': 'MyTorLock'
    },
    'freshon.tv': {
        'tv_keywords': '{title} s{season:2}e{episode:2}'
    }
}
```

3.3 Adding providers

Adding a custom provider is similar to using overrides, although you'll be using a JSON file, per provider or with all your custom providers, unless you add them all in your custom *overrides.py*, which also works.

To add a provider, simply create a file with the `.json` extension under the `providers` folder in your userdata, ie. as `~/.kodi/userdata/addon_data/script.elementum.burst/providers/nice_provider.json`, and make sure it follows the format below (hopefully with `"subpage": false`):

```
{
    "1337x": {
        "anime_extra": "",
        "anime_keywords": "{title} {episode}",
        "anime_query": "EXTRA",
        "base_url": "http://www.1337x.to/search/QUERY/1/",
        "color": "FFF14E13",
        "general_extra": "",
        "general_keywords": "{title}",
        "general_query": "EXTRA",
        "language": null,
        "login_failed": "",
```

```
        "login_object": "",
        "login_path": null,
        "movie_extra": "",
        "movie_keywords": "{title} {year}",
        "movie_query": "EXTRA",
        "name": "1337x",
        "parser": {
            "infohash": "",
            "name": "item('a', order=2)",
            "peers": "item(tag='td', order=3)",
            "row": "find_once(tag='body').find_all('tr')",
            "seeds": "item(tag='td', order=2)",
            "size": "item(tag='td', order=5)",
            "torrent": "item(tag='a', attribute='href', order=2)"
        },
        "private": false,
        "season_extra": "",
        "season_extra2": "",
        "season_keywords": "{title} Season {season:2}",
        "season_keywords2": "{title} Season{season}",
        "season_query": "EXTRA",
        "separator": "+",
        "show_query": "",
        "subpage": true,
        "tv_extra": "",
        "tv_extra2": "",
        "tv_keywords": "{title} s{season:2}e{episode:2}",
        "tv_keywords2": ""
    }
}
```

3.3.1 Provider fields

name

The provider's name as displayed to the user, typically with color.

color

The color of the provider name using Kodi's ARGB (alpha-red-green-blue) color format.

base_url

The `base_url` is the part of the provider's URL that is **always** found in your browser bar when you visit or more importantly, search the site. It may or may not contain the `QUERY` part (more on that later); it really only depends on the **common** part of the different search queries.

language

Forces a language preference for translations if they're available, eg. `es`

private

Boolean flag to mark this provider as private, see *PrivateProviders*.

separator

Space separator used in URL queries by this provider, typically `%20` for an encoded white-space or `+`

subpage

The most expensive boolean flag, to be avoided as much as possible. This tells Burst that we have no choice but to open **each and every** link to get to the torrent or magnet link. As it stands, we also waste the `torrent` (more on that later) definition under `parser`, which becomes the link to follow, and the page at that link gets automatically processed to find a magnet or torrent link in it.

*_query

Second part of the URL after `base_url` which will contain the `QUERY` keyword if it's not already in the `base_url`. This typically include category parameters specific to each provider, ie. `/movies/QUERY`

*_extra

The most confusing part of queries. Those will contain *extra* parameters, typically categories also, replacing the `EXTRA` keyword often found in the respective `*_query` definition, and often simply for the convenience of shorter `*_query` definitions. Note that this is mostly always just an empty string and not being used.

*_keywords

Keyword definitions for the different search types, with special placeholders like `{title}` for a movie or TV show title.

List of keyword types

- `{title}` Movie or TV show title
- `{year}` Release date, typically for movies only
- `{season}` Season number. Using `{season:2}` pads to 2 characters with leading zeros, eg. `s{season:2}` would become `s01` for an episode of season 1.
- `{episode}` Episode number, same formatting as `{season}` with regards to padding, ie. `{episode:2}`. Typically used with season as such: `s{season:2}e{episode:2}`

parser

This is the most important part of every provider, and tells Burst how to find torrents within search result pages. The first parser definition to be used is the `row`, and is also the "parent" to all to the others. It most usually ends with a `find_all('tr')`, and tells Burst which HTML tags, typically table rows, hold the results we're interested in. All other parser definitions will then look **within** each row for their respective information. Each other parser definition

tells Burst what HTML tag has its information, for example `item(tag='td', order=1)` for `name` tells Burst that the torrent name is in the first table column of each row.

TODO: A more detailed description of parser fields and a tutorial on how to actually create providers will soon be added.

3.3.2 Private providers

login_path

The `login_path` is the part of the URL used for logging in, typically something like `"/login.php"`. This can be found by inspecting the login form's HTML and taking its `action` attribute.

login_object

The `login_object` represents the form elements sent to the `login_path`. For built-in private providers, placeholders are used to replace setting values for the username and password (`USERNAME` and `PASSWORD` respectively). Custom providers cannot define new settings, and must therefore put the username and password in the `login_object` directly.

login_failed

String that must **not** be included in the response's content. If this string is present in the page when trying to login, it returns as having failed and no search queries will be sent.

3.4 burst package

3.4.1 Subpackages

burst.parser package

Submodules

burst.parser.HTMLParser module

A parser for HTML and XHTML.

exception `burst.parser.HTMLParser.`**`HTMLParseError`**(*msg*, *position=(None*, *None)*)

Bases: `exceptions.Exception`

Exception raised for all parse errors.

class `burst.parser.HTMLParser.`**`HTMLParser`**

Bases: *`burst.parser.markupbase.ParserBase`*

Find tags and other markup and call handler functions.

Usage: p = HTMLParser() p.feed(data) ... p.close()

Start tags are handled by calling self.handle_starttag() or self.handle_startendtag(); end tags by self.handle_endtag(). The data between tags is passed from the parser to the derived class by calling self.handle_data() with the data as argument (the data may be split up in arbitrary chunks). Entity references are

passed by calling self.handle_entityref() with the entity reference as the argument. Numeric character references are passed to self.handle_charref() with the string containing the reference as the argument.

CDATA_CONTENT_ELEMENTS = ('script', 'style')

reset()
Reset this instance. Loses all unprocessed data.

feed(*data*)
Feed data to the parser.

Call this as often as you want, with as little or as much text as you want (may include 'n').

close()
Handle any buffered data.

error(*message*)

get_starttag_text()
Return full source of start tag: '<...>'.

set_cdata_mode(*elem*)

clear_cdata_mode()

goahead(*end*)

parse_html_declaration(*i*)

parse_bogus_comment(*i*, *report=1*)

parse_pi(*i*)

parse_starttag(*i*)

check_for_whole_start_tag(*i*)

parse_endtag(*i*)

handle_startendtag(*tag*, *attrs*)

handle_starttag(*tag*, *attrs*)

handle_endtag(*tag*)

handle_charref(*name*)

handle_entityref(*name*)

handle_data(*data*)

handle_comment(*data*)

handle_decl(*decl*)

handle_pi(*data*)

unknown_decl(*data*)

entitydefs = None

unescape(*s*)

burst.parser.ehp module

” All the credit of this code to Iury de oliveira gomes figueiredo Easy Html Parser is an AST generator for html/xml documents. You can easily delete/insert/extract tags in html/xml documents as well as look for patterns. https://github.com/iogf/ehp

class burst.parser.ehp.**Attribute**
Bases: dict

This class holds the tags's attributes. The idea consists in providing an efficient and flexible way of manipulating tags attributes inside the dom.

Example: dom = Html().feed(‘ foo ’)

for ind in dom.sail(): if ind.name == ‘p’: ind.attr[’style’] = “color:blue”

It would change to color blue.

class burst.parser.ehp.**Root**(*name=None*, *attr=None*)
Bases: list

A Root instance is the outmost node for a xml/html document. All xml/html entities inherit from this class.

html = Html() dom = html.feed(‘<html> ... </body>’)

dom.name == ‘’ True type(dom) == Root True

sail()
This is used to navigate through the xml/html document. Every xml/html object is represented by a python class instance that inherits from Root.

The method sail is used to return an iterator for these objects.

Example: data = ‘<a>   ’

html = Html() dom = html.feed(data)

for ind in dom.sail(): print type(ind),’,’, ind.name

It would output.

<class ‘ehp.Root’> , a <class ‘ehp.Root’> , b

index(*item*, ***kwargs*)
This is similar to index but uses id to check for equality.

Example:

data = ‘<a>’ html = Html() dom = html.feed(data)

for root, ind in dom.sail_with_root(): print root.name, ind.name, root.index(ind)

It would print.

a b 0 a b 1 a 0

The line where it appears ‘ a 0’ corresponds to the outmost object. The outmost object is an instance of Root that contains all the other objects. :param item:

remove(*item*)
This is as list.remove but works with id.

data = ‘<a>’ html = Html() dom = html.feed(data) for root, ind in dom.sail_with_root(): if ind.name == ‘b’: root.remove(ind)

print dom

It should print.

<a >

find(*name=''*, *every=1*, *start=1*, **args*)
It is used to find all objects that match name.

Example 1:

data = '<a>' html = Html() dom = html.feed(data)

for ind in dom.find('b'): print ind

It should print.

Example 2.

data = '<body> alpha.   beta. </body>' html = Html() dom = html.feed(data)

for ind in dom.find('p', ('style', 'color:green')): print ind

Or

for ind in dom.find('p', ('style', ['color:green', 'color:red'])): print ind

Output.

 beta.

find_once(*tag=None*, *select=None*, *order=1*)
" It returns the nth (order) ocurrence from the tag matching with the attributes from select

find_all(*tag=None*, *select=None*, *every=1*, *start=1*)
" It returns all ocurrences from the tag matching with the attributes from select

find_with_root(*name*, **args*)
Like Root.find but returns its parent tag.

from ehp import *

html = Html() dom = html.feed(''<body> alpha   beta </body>'')

for root, ind in dom.find_with_root('p'): root.remove(ind)

print dom

It would output.

<body > </body>

by_id(*id_value*)
It is a shortcut for finding an object whose attribute 'id' matches id.

Example:

data = '<a>' html = Html() dom = html.feed(data)

print dom.byid('foo') print dom.byid('bar')

It should print.

 None

take (**args*)
It returns the first object whose one of its attributes matches (key0, value0), (key1, value1), ...

Example:

data = '<a>' html = Html() dom = html.feed(data)

print dom.take(('id', 'foo')) print dom.take(('id', 'foo'), ('size', '2'))

take_with_root (**args*)
Like Root.take but returns the tag parent.

match (**args*)
It returns a sequence of objects whose attributes match. (key0, value0), (key1, value1), ...

Example:

data = '' html = Html() dom = html.feed(data)

for ind in dom.match(('size', '1')): print ind

It would print.

match_with_root (**args*)
Like Root.match but with its parent tag.

Example:

from ehp import *

html = Html() dom = html.feed(''<body> xxx   mmm </body>'')

for root, ind in dom.match_with_root(('style', 'color:black')): del ind.attr['style']

item = dom.fst('body') item.attr['style'] = 'color:black'

print dom

Output.

<body style="color:black" > xxx   mmm </body>

join (*delim*, **args*)
It joins all the objects whose name appears in args.

Example 1:

html = Html() data = '<a> This is cool. That is. ' dom = html.feed(data)

print dom.join('', 'b') print type(dom.join('b'))

It would print.

 This is cool. That is. <type 'str'>

Example 2:

html = Html() data = '<a> alpha<c>beta</c> gamma' dom = html.feed(data)

print dom.join('', 'b', 'c')

It would print.

 alpha<c >beta</c>gamma

Example 3:

html = Html() data = '<a>alpha<c>beta</c>gamma' dom = html.feed(data)

print dom.join('n', DATA)

It would print.

alpha beta gamma

fst (*name*, **args*)
It returns the first object whose name matches.

Example 1:

html = Html() data = '<body> Cool. </body>' dom = html.feed(data)

print dom.fst('em')

It outputs.

 Cool.

Example 2:

data = '<body> alpha.   beta. </body>' html = Html() dom = html.feed(data)

for ind in dom.find('p', ('style', 'color:green')): print ind

print dom.fst('p', ('style', 'color:green')) print dom.fst_with_root('p', ('style', 'color:green'))

Output:

 beta. beta. (<ehp.Tag object at 0xb7216c0c>, <ehp.Tag object at 0xb7216d24>)

fst_with_root (*name*, **args*)
Like fst but returns its item parent.

Example:

html = Html() data = '<body> Cool. </body>' dom = html.feed(data)

root, item dom.fst_with_root('em') root.insert_after(item, Tag('p')) print root

It outputs.

<body > Cool. </body>

For another similar example, see help(Root.fst)

text ()
It returns all objects whose name matches DATA. It basically returns a string corresponding to all asci characters that are inside a xml/html tag.

Example:

html = Html() data = '<body>This is all the text.</body>' dom = html.feed(data)

print dom.fst('em').text()

It outputs.

This is all the text.

Notice that if you call text() on an item with children then it returns all the *printable* characters for that node.

write (*filename*)
It saves the structure to a file.

sail_with_root()
This one works like sail(), however it yields the tag's parents as well as the child tag.

For an example, see help(Root.remove).

walk()
Like sail but carries name and attr.

Example:

html = Html() data = '<body> This is all the text.</body>' dom = html.feed(data)

for ind, name, attr in dom.walk(): print 'TAG:', ind print 'NAME:', name print 'ATTR:', attr

It should print.

TAG: NAME: 1 ATTR: TAG: This is all the text. NAME: 1 ATTR: TAG: This is all the text. NAME: em ATTR: TAG: <body > This is all the text.</body> NAME: body ATTR:

walk_with_root()
Like walk but carries root.

Example:

html = Html() data = '<body>alpha</body>' dom = html.feed(data)

for (root, name, attr), (ind, name, attr) in dom.walk_with_root(): print root, name, ind, name

Output:

alpha 1 alpha 1 <body >alpha</body> em alpha em <body >alpha</body> body <body >alpha</body> body

insert_after(*y*, *k*)
Insert after a given tag.

For an example, see help(Root.fst_with_root).

insert_before(*y*, *k*)
Insert before a given tag.

For a similar example, see help(Root.fst_with_root).

parent(*dom*)
Find the parent tag

list_(*text=''*)

select(*text=''*)

get_attributes(*text*)

class burst.parser.ehp.**Tag**(*name*, *attr=None*)
Bases: *burst.parser.ehp.Root*

This class's instances represent xml/html tags under the form: <name key="value" ...> ... </name>.

It holds useful methods for parsing xml/html documents.

class burst.parser.ehp.**Data**(*data*)
Bases: *burst.parser.ehp.Root*

The pythonic representation of data that is inside xml/html documents.

All data that is not a xml/html token is represented by this class in the structure of the document.

Example:

html = Html() data = '<body>alpha</body>' dom = html.feed(data)

x = dom.fst('em')

x holds a Data instance.

type(x[0]) print x[0]

Output:

<class 'ehp.Data'> alpha

The Data instances are everywhere in the document, when the tokenizer finds them between the xml/html tags it builds up the structure identically to the document.

text()

class burst.parser.ehp.**XTag**(*name*, *attr=None*)
Bases: *burst.parser.ehp.Root*

This tag is the representation of html's tags in XHTML style like It is tags which do not have children.

class burst.parser.ehp.**Meta**(*data*)
Bases: *burst.parser.ehp.Root*

class burst.parser.ehp.**Code**(*data*)
Bases: *burst.parser.ehp.Root*

class burst.parser.ehp.**Amp**(*data*)
Bases: *burst.parser.ehp.Root*

class burst.parser.ehp.**Pi**(*data*)
Bases: *burst.parser.ehp.Root*

class burst.parser.ehp.**Comment**(*data*)
Bases: *burst.parser.ehp.Root*

class burst.parser.ehp.**Tree**
Bases: object

The engine class.

clear()
Clear the outmost and stack for a new parsing.

last()
Return the last pointer which point to the actual tag scope.

nest(*name*, *attr*)
Nest a given tag at the bottom of the tree using the last stack's pointer.

dnest(*data*)
Nest the actual data onto the tree.

xnest(*name*, *attr*)
Nest a XTag onto the tree.

ynest(*data*)

mnest(*data*)

cnest(*data*)

rnest(*data*)

inest(*data*)

enclose(*name*)
When found a closing tag then pops the pointer's scope from the stack so pointing to the earlier scope's tag.

class burst.parser.ehp.**Html**
Bases: *burst.parser.HTMLParser.HTMLParser*

The tokenizer class.

fromfile(*filename*)
It builds a structure from a file.

feed(*data*)

handle_starttag(*name*, *attr*)
When found an opening tag then nest it onto the tree

handle_startendtag(*name*, *attr*)
When found a XHTML tag style then nest it up to the tree

handle_endtag(*name*)
When found a closing tag then makes it point to the right scope

handle_data(*data*)
Nest data onto the tree.

handle_decl(*decl*)

unknown_decl(*decl*)

handle_charref(*data*)

handle_entityref(*data*)

handle_pi(*data*)

handle_comment(*data*)

burst.parser.markupbase module

Shared support for scanning document type declarations in HTML and XHTML.

This module is used as a foundation for the HTMLParser and sgmllib modules (indirectly, for htmllib as well). It has no documented public API and should not be used directly.

class burst.parser.markupbase.**ParserBase**
Parser base class which provides some common support methods used by the SGML/HTML and XHTML parsers.

error(*message*)

reset()

getpos()
Return current line number and offset.

updatepos(*i*, *j*)

parse_declaration(*i*)

parse_marked_section(*i*, *report=1*)

parse_comment(*i*, *report=1*)

unknown_decl(*data*)

burst.providers package

Submodules

burst.providers.burst_overrides module

Default Burst overrides

burst.providers.burst_overrides.**overrides**
Default overrides dictionary

burst.providers.burst_overrides.**source**()
See source

Note: This just a dummy method for documentation

burst.providers.definitions module

Definitions and overrides loader

burst.providers.definitions.**load_providers**(*path*, *custom=False*, *fix_seasons=False*)
Definitions loader for json files

Parameters

- **path** (*str*) – Path to json file to be loaded
- **custom** (*bool*) – Boolean flag to specify if this is a custom provider
- **fix_seasons** (*bool*) – Boolean flag to apply default fix to seasons keywords

burst.providers.definitions.**load_overrides**(*path*, *custom=False*)
Overrides loader for Python files

Note: Overrides must be in an `overrides` dictionary.

Parameters

- **path** (*str*) – Path to Python file to be loaded
- **custom** (*bool*) – Boolean flag to specify if this is a custom overrides file

burst.providers.definitions.**update_definitions**(*provider*, *definition*, *custom=False*, *fix_seasons=False*)
Updates global definitions with a single provider's definitions

Parameters

- **provider** (*str*) – Provider ID
- **definition** (*dict*) – Loaded provider's definitions to be merged with the global definitions
- **custom** (*bool*) – Boolean flag to specify if this is a custom provider
- **fix_seasons** (*bool*) – Boolean flag to apply default fix to seasons keywords

burst.providers.definitions.**update**(*d*, *u*)
Utility method to recursively merge dictionary values of definitions

Parameters

- **d** (*dict*) – Current provider definitions
- **u** (*dict*) – Dictionary of definitions to be updated

burst.providers.helpers module

Helpers for providers that need special filtering

burst.providers.helpers.**t411season**(*season*)

burst.providers.helpers.**t411episode**(*episode*)

3.4.2 Submodules

3.4.3 burst.burst module

Burst processing thread

burst.burst.**search**(*payload*, *method='general'*)
Main search entrypoint

Parameters

- **payload** (*dict*) – Search payload from Elementum.
- **method** (*str*) – Type of search, can be `general`, `movie`, `show`, `season` or `anime`

Returns All filtered results in the format Elementum expects

Return type list

burst.burst.**got_results**(*provider*, *results*)
Results callback once a provider found all its results, or not

Parameters

- **provider** (*str*) – The provider ID
- **results** (*list*) – The list of results

burst.burst.**extract_torrents**(*provider*, *client*)
Main torrent extraction generator for non-API based providers

Parameters

- **provider** (*str*) – Provider ID
- **client** (*Client*) – Client class instance

Yields *tuple* – A torrent result

burst.burst.**extract_from_api**(*provider*, *client*)
Main API parsing generator for API-based providers

An almost clever API parser, mostly just for YTS, RARBG and T411

Parameters

- **provider** (*str*) – Provider ID

- **client** (Client) – Client class instance

Yields *tuple* – A torrent result

burst.burst.**extract_from_page**(*provider*, *content*)
Sub-page extraction method

Parameters

- **provider** (*str*) – Provider ID
- **content** (*str*) – Page content from Client instance

Returns Torrent or magnet link extracted from sub-page

Return type str

burst.burst.**run_provider**(*provider*, *payload*, *method*)
Provider thread entrypoint

Parameters

- **provider** (*str*) – Provider ID
- **payload** (*dict*) – Search payload from Elementum
- **method** (*str*) – Type of search, can be `general`, `movie`, `show`, `season` or `anime`

3.4.4 burst.client module

Burst web client

class burst.client.**Client**
Web client class with automatic charset detection and decoding

cookies()
Saved client cookies

Returns A list of saved Cookie objects

Return type list

open(*url*, *language='en'*, *post_data=None*, *get_data=None*)
Opens a connection to a webpage and saves its HTML content in `self.content`

Parameters

- **url** (*str*) – The URL to open
- **language** (*str*) – The language code for the `Content-Language` header
- **post_data** (*dict*) – POST data for the request
- **get_data** (*dict*) – GET data for the request

login(*url*, *data*, *fails_with*)
Login wrapper around `open`

Parameters

- **url** (*str*) – The URL to open
- **data** (*dict*) – POST login data
- **fails_with** (*str*) – String that must **not** be included in the response's content

Returns Whether or not login was successful

Return type bool

burst.client.**get_cloudhole_key**()
CloudHole API key fetcher

Returns A CloudHole API key

Return type str

burst.client.**get_cloudhole_clearance**(*cloudhole_key*)
CloudHole clearance fetcher

Parameters **cloudhole_key** (*str*) – The CloudHole API key saved in settings or from get_cloudhole_key directly

Returns A CloudHole clearance cookie and user-agent string

Return type tuple

3.4.5 burst.filtering module

Burst filtering class and methods

class burst.filtering.**Filtering**
Filtering class

resolutions
OrderedDict – Ordered dictionary of resolution filters to be used depending on settings

resolutions_allow
list – List of resolutions to allow in search results

release_types
dict – Dictionary of release types to be used depending on settings

releases_allow
list – List of release types to allow in search results

releases_deny
list – List of release types to deny in search results

require_keywords
list – List of keywords to require in search results

min_size
float – Minimum required size

max_size
float – Maximum possible size

filter_title
bool – Whether or not this provider needs titles to be double-checked, typically used for providers that return too many results from their search engine when no results are found (ie. TorLock and TorrentZ)

queries
list – List of queries to be filtered

extras
list – List of extras to be filtered

info
dict – Payload from Elementum

kodi_language
str – Language code from Kodi if kodi_language setting is enabled

language_exceptions
list – List of providers for which not to apply `kodi_language` setting

url
str – URL of this filtering request

get_data
dict – GET data for client request

post_data
dict – POST data for client request

title
str – Result title to be used when matching with `filter_title` enabled

reason
str – Rejection reason when result does not match

results
list – Filtered, accepted results

use_general(*provider*, *payload*)
Setup method to define general search parameters

Parameters

- **provider** (*str*) – Provider ID
- **payload** (*dict*) – Elementum search payload

use_movie(*provider*, *payload*)
Setup method to define movie search parameters

Parameters

- **provider** (*str*) – Provider ID
- **payload** (*dict*) – Elementum search payload

use_episode(*provider*, *payload*)
Setup method to define episode search parameters

Parameters

- **provider** (*str*) – Provider ID
- **payload** (*dict*) – Elementum search payload

use_season(*provider*, *info*)
Setup method to define season search parameters

Parameters

- **provider** (*str*) – Provider ID
- **payload** (*dict*) – Elementum search payload

use_anime(*provider*, *info*)
Setup method to define anime search parameters

Parameters

- **provider** (*str*) – Provider ID

- **payload** (*dict*) – Elementum search payload

information(*provider*)
Debugging method to print keywords and file sizes

check_sizes()
Internal method to make sure size range settings are valid

read_keywords(*keywords*)
Create list from keywords where the values are marked between curly brackets, ie. {title}

Parameters **keywords** (*str*) – String with all the keywords, ie. '{title} {year} movie'

Returns List of keywords, ie. ['{title}', '{year}']

Return type list

process_keywords(*provider*, *text*)
Processes the query payload from a provider's keyword definitions

Parameters

- **provider** (*str*) – Provider ID
- **text** (*str*) – Keyword placeholders from definitions, ie. {title}

Returns Processed query keywords

Return type str

verify(*provider*, *name*, *size*)
Main filtering method to match torrent names, resolutions, release types and size filters

Parameters

- **provider** (*str*) – Provider ID
- **name** (*str*) – Torrent name
- **size** (*str*) – Arbitrary torrent size to be parsed

Returns `True` if torrent name passed filtering, `False` otherwise.

Return type bool

in_size_range(*size*)
Compares size ranges

Parameters **size** (*str*) – File size string, ie. `1.21 GB`

Returns `True` if file size is within desired range, `False` otherwise

Return type bool

determine_resolution(*name*)
Determine torrent resolution from defined filters. Defaults to `filter_480p`.

Parameters **name** (*str*) – Name of the torrent to determine the resolution for

Returns The filter key of the determined resolution, see self.resolutions

Return type str

normalize_name(*value*)
Method to normalize strings

Replaces punctuation with spaces, unquotes and unescapes HTML characters.

Parameters **value** (*str*) – File name or directory string to convert

Returns Converted file name or directory string

Return type str

included(*value*, *keys*, *strict=False*)
Check if the keys are present in the string

Parameters

- **value** (*str*) – Name of the torrent to check
- **keys** (*list*) – List of strings that must be included in `value`
- **strict** (*bool*) – Boolean flag to accept or not partial results

Returns True if any (or all if `strict`) keys are included, False otherwise.

Return type bool

unescape(*name*)

Unescapes all HTML entities from a string using HTMLParser().unescape()

Parameters **name** (*str*) – String to convert

Returns Converted string

Return type str

exception(*title=None*)
Change the title to the standard name in torrent sites

Parameters **title** (*str*) – Title to check

Returns Standard title

Return type str

burst.filtering.**apply_filters**(*results_list*)
Applies final result de-duplicating, hashing and sorting

Parameters **results_list** (*list*) – Formatted results in any order

Returns Filtered and sorted results

Return type list

burst.filtering.**cleanup_results**(*results_list*)
Remove duplicate results, hash results without an info_hash, and sort by seeders

Parameters **results_list** (*list*) – Results to clean-up

Returns De-duplicated, hashed and sorted results

Return type list

3.4.6 burst.ordereddict module

class burst.ordereddict.**OrderedDict**(**args*, ***kwds*)
Bases: dict, UserDict.DictMixin

Backport of `collections.OrderedDict` for Python 2.6 (Kodi 16)

clear()

popitem(*last=True*)

keys()

setdefault(*key*, *default=None*)

update(*other=None*, ***kwargs*)

pop(*key*, **args*)

values()

items()

iterkeys()

itervalues()

iteritems()

copy()

classmethod fromkeys(*iterable*, *value=None*)

3.4.7 burst.provider module

Provider thread methods

burst.provider.**generate_payload**(*provider*, *generator*, *filtering*, *verify_name=True*, *verify_size=True*)

Payload formatter to format results the way Elementum expects them

Parameters

- **provider** (*str*) – Provider ID
- **generator** (*function*) – Generator method, can be either `extract_torrents` or `extract_from_api`
- **filtering** (`Filtering`) – Filtering class instance
- **verify_name** (*bool*) – Whether to double-check the results' names match the query or not
- **verify_size** (*bool*) – Whether to check the results' file sizes

Returns Formatted results

Return type list

burst.provider.**process**(*provider*, *generator*, *filtering*, *verify_name=True*, *verify_size=True*)

Method for processing provider results using its generator and Filtering class instance

Parameters

- **provider** (*str*) – Provider ID
- **generator** (*function*) – Generator method, can be either `extract_torrents` or `extract_from_api`
- **filtering** (`Filtering`) – Filtering class instance
- **verify_name** (*bool*) – Whether to double-check the results' names match the query or not
- **verify_size** (*bool*) – Whether to check the results' file sizes

3.4.8 burst.utils module

Burst utilities

class burst.utils.**Magnet**(*magnet*)
Magnet link parsing class

Parameters **magnet** (*str*) – A magnet link string

info_hash
str – Info-hash from the magnet link

name
str – Name of torrent

trackers
list – List of trackers in magnet link

burst.utils.**get_domain**(*url*)

burst.utils.**get_alias**(*definition*, *alias*)

burst.utils.**get_providers**()
Utility method to get all provider IDs available in the definitions

Returns All available provider IDs

Return type list

burst.utils.**get_enabled_providers**()
Utility method to get all enabled provider IDs

Returns All available enabled provider IDs

Return type list

burst.utils.**get_icon_path**()
Utility method to Burst's icon path

Returns Path to Burst's icon

Return type str

burst.utils.**translation**(*id_value*)
Utility method to get translations

Parameters **id_value** (*int*) – Code of translation to get

Returns Translated string

Return type str

burst.utils.**get_int**(*string*)
Utility method to convert a number contained in a string to an integer

Parameters **string** (*str*) – Number contained in a string

Returns The number as an integer, or 0

Return type int

burst.utils.**get_float**(*string*)
Utility method to convert a number contained in a string to a float

Parameters **string** (*str*) – Number contained in a string

Returns The number as a float, or 0.0

Return type float

burst.utils.**size_int**(*size_txt*)
Utility method to convert a file size contained in a string to an integer of bytes

Parameters **string** (*str*) – File size with suffix contained in a string, eg. `1.21 GB`

Returns The number of bytes as a float, or 0.0

Return type float

burst.utils.**clean_number**(*string*)
Utility method to clean up a number contained in a string to dot decimal format

Parameters **string** (*str*) – Number contained in a string

Returns The formatted number as a string

Return type str

burst.utils.**clean_size**(*string*)
Utility method to remove unnecessary information from a file size string, eg. '6.5 GBytes' -> '6.5 GB'

Parameters **string** (*str*) – File size string to clean up

Returns Cleaned up file size

Return type str

burst.utils.**sizeof**(*num*, *suffix='B'*)
Utility method to convert a file size in bytes to a human-readable format

Parameters

- **num** (*int*) – Number of bytes
- **suffix** (*str*) – Suffix for 'bytes'

Returns The formatted file size as a string, eg. `1.21 GB`

Return type str

burst.utils.**notify**(*message*, *image=None*)
Creates a notification dialog

Parameters

- **message** (*str*) – The message to show in the dialog
- **image** (*str*) – Path to an icon for this dialog

burst.utils.**clear_cache**()
Clears cookies from Burst's cache

burst.utils.**encode_dict**(*dict_in*)
Encodes dict values to UTF-8

Parameters **dict_in** (*dict*) – Input dictionary with unicode values

Returns Output dictionary with UTF-8 encoded values

Return type dict

burst.utils.**iri2uri**(*iri*)

burst.utils.**encode**(*c*)

- modindex

CHAPTER 4

Credits

- @mancuniancol for all his work on Magnetic, this add-on wouldn't have been possible without him.
- All the alpha and beta testers that led to the first stable release.

Python Module Index

b

Index

G

H

I

J

K

L

M

N

W

X

Y