

azuracast_xmltv Documentation

Contents

Module azuracast_xmltv	1
Functions	1
Function clean_text	1
Function create_gap_fillers	2
Function create_m3u	2
Function create_m3u_all	2
Function create_rss_station	2
Function create_xmltv	2
Function errprint	3
Function get_programme	3
Function get_stations	3
Function handle_instance	3
Function is_video	3
Function linkify_urls	4
Function main	4
Function replace_vars	4
Function rfc3339_date	4
Function sanitize	4
Function urljoin	4
Function valid_path	5
Function valid_url	5
Function valid_url_path	5
Classes	5
Class AzuraCastAPI	5
Methods	5
Class CustomFormatter	6
Ancestors (in MRO)	6
Class Range	6
Ancestors (in MRO)	7

Module azuracast_xmltv

azuracast_xmltv

Fetches EPG info from an AzuraCast (web radio) instance and provides XMLTV Tuner M3U, EPG XML, and RSS feed listings.

Functions

Function clean_text

```
def clean_text(
    description
)
```

Remove all leftover whitespace from a programme description.

Function create_gap_fillers

```
def create_gap_fillers(  
    station,  
    programmes  
)
```

Find gaps between programmes, return list of “gap filler” entries.

Function create_m3u

```
def create_m3u(  
    instance_url,  
    station,  
    groups,  
    output_folder='',  
    add_tvg_url=False,  
    add_radio_tag=False  
)
```

Create an M3U playlist for a single station.

Function create_m3u_all

```
def create_m3u_all(  
    instance_url,  
    stations,  
    groups,  
    output_folder='',  
    add_tvg_url=False,  
    add_radio_tag=False  
)
```

Create an M3U playlist for all stations.

Function create_rss_station

```
def create_rss_station(  
    instance_url,  
    station,  
    programmes,  
    output_folder='')  
)
```

Create an RSS Feed for a single station.

Function create_xmltv

```
def create_xmltv(  
    instance_url,  
    stations,  
    programmes,  
    output_folder='',  
    add_gzip=False  
)
```

Create an XMLTV EPG for all stations.

Function errprint

```
def errprint(  
    *args,  
    **kwargs  
)
```

Print error messages to stderr, nicely.

Function get_programme

```
def get_programme(  
    api,  
    station,  
    num_days=7,  
    fill_gaps=False  
)
```

Get a channel's programme.

Function get_stations

```
def get_stations(  
    api,  
    channel_icon_url='',  
    public_only=True,  
    videostream_keywords=[]  
)
```

Get station info from this instance.

Function handle_instance

```
def handle_instance(  
    instance_url='https://demo.azuracast.com',  
    api_key=None,  
    public_only=False,  
    output_folder='',  
    make_m3u=False,  
    make_rss=False,  
    channel_icon_url=None,  
    custom_player_url=None,  
    num_days=7,  
    fill_gaps=False,  
    add_radio_tag=True,  
    add_tvg_url=False,  
    add_gzip=True,  
    m3u_group_title=''  
)
```

Handle one AzuraCast instance.

Function is_video

```
def is_video(  
    name,  
    keywords  
)
```

Determine if a stream is a video stream.
Check its name against a list of keywords.

Function linkify_urls

```
def linkify_urls(  
    text  
)
```

(Try to) replace any URLs in text with anchor links, for HTML conversion.

Based on John Gruber's article: https://daringfireball.net/2010/07/improved_regex_for_matching_urls and Gist: <https://gist.github.com/gruber/249502> "Liberal Regex Pattern for Any URLs"

Function main

```
def main()
```

Main (runs if executed directly).

Function replace_vars

```
def replace_vars(  
    dictionary,  
    text  
)
```

Replace moustache-type variables in text (one level deep).

Function rfc3339_date

```
def rfc3339_date(  
    dt,  
    utc_z=False  
)
```

Return datetime object as RFC 3339 string.

RFC 3339 is an ISO 8601 profile, compatible with W3C-DTF format.

utc_z=True outputs Z instead of +00:00 for UTC datetimes.

Function sanitize

```
def sanitize(  
    name  
)
```

Sanitize a station shortname to become RFC2838-compliant.

Function urljoin

```
def urljoin(  
    *args  
)
```

Safely join parts of an URL.

TODO: Fix second slash being removed from scheme like "https://".

Function `valid_path`

```
def valid_path(  
    arg  
)
```

Check if argument is a valid output folder path.

Function `valid_url`

```
def valid_url(  
    arg  
)
```

Check if argument is a valid URL (scheme + domain).

Function `valid_url_path`

```
def valid_url_path(  
    arg  
)
```

Check if argument is a valid URL (scheme + domain + path).

Classes**Class** `AzuraCastAPI`

```
class AzuraCastAPI(  
    instance_url,  
    api_key='',  
    timeout=30  
)
```

AzuraCast API interface, strictly JSON.

Initialize AzuraCastAPI instance

Methods**Method** `fail`

```
def fail(  
    self  
)
```

Return True if last API access failed.

Method `get`

```
def get(  
    self,  
    endpoint,  
    params={},  
    quit_on_error=False  
)
```

Get API JSON response; return empty list on error (or quit).

Method `verify`

```
def verify(  
    self  
)
```

Verify we're talking to an AzuraCast server.

Check the /status API endpoint and online state.

Warns user when redirected (possibly wrong URLs in output).

Class `CustomFormatter`

```
class CustomFormatter(  
    prog,  
    indent_increment=2,  
    max_help_position=24,  
    width=None  
)
```

Format output of help text nicely.

This works like `ArgumentDefaultsHelpFormatter` plus `RawDescriptionHelpFormatter` in one: It wraps arguments, prolog and epilog nicely at terminal width and also keeps newlines in prolog and epilog intact.

Ancestors (in MRO)

- [argparse.ArgumentDefaultsHelpFormatter](#)
- [argparse.HelpFormatter](#)

Class `Range`

```
class Range(  
    minimum=None,  
    maximum=None,  
    *args,  
    **kwargs  
)
```

Information about how to convert command line strings to Python objects.

Action objects are used by an `ArgumentParser` to represent the information needed to parse a single argument from one or more strings from the command line. The keyword arguments to the Action constructor are also all attributes of Action instances.

Keyword Arguments:

- `option_strings` -- A list of command-line option strings which should be associated with this action.
- `dest` -- The name of the attribute to hold the created object(s)
- `nargs` -- The number of command-line arguments that should be consumed. By default, one argument will be consumed and a single value will be produced. Other values include:
 - `N` (an integer) consumes `N` arguments (and produces a list)
 - `'?'` consumes zero or one arguments
 - `'*'` consumes zero or more arguments (and produces a list)
 - `'+'` consumes one or more arguments (and produces a list)

Note that the difference between the default and `nargs=1` is that with the default, a single value will be produced, while with

- nargs=1, a list containing a single value will be produced.
- const -- The value to be produced if the option is specified and the option uses an action that takes no values.
 - default -- The value to be produced if the option is not specified.
 - type -- A callable that accepts a single string argument, and returns the converted value. The standard Python types str, int, float, and complex are useful examples of such callables. If None, str is used.
 - choices -- A container of values that should be allowed. If not None, after a command-line argument has been converted to the appropriate type, an exception will be raised if it is not a member of this collection.
 - required -- True if the action must always be specified at the command line. This is only meaningful for optional command-line arguments.
 - help -- The help string describing the argument.
 - metavar -- The name to be used for the option's argument with the help string. If None, the 'dest' value will be used as the name.

Ancestors (in MRO)

- [argparse.Action](#)
- [argparse._AttributeHolder](#)

Generated by *pdoc* 0.10.0 (<https://pdoc3.github.io>).