Estructura y funcionamiento de AlfaChannelHelper (AH)

Este módulo estás basado en el desarrollo inicial de **Delta** del Alfa Development Group.

Es preocupante la mantenibilidad de Alfa, cada día con menos recursos. Con este sistema el 80% del código será común a todos los canales que usen la clase, dejando gran libertad al desarrollador del canal para imponer su estilo. Si con el tiempo lográramos convertir todos los canales, su mantenimiento pasaría a ser algo banal, pudiéndose encargar cualquiera de cualquier canal si su desarrollador no está disponible. El tiempo libre que nos deja lo podríamos usar en desarrollos más creativos y gratificantes.

Este concepto se basa en tres piezas fundamentales:

- **Diccionario "finds",** con estructura común para todos los canales, donde se sitúan la expresiones regulares (**en formato BeatifulSoup (BS)** y algunas en regex) y parámetros que se pasan a AH para el procesamiento personalizado del canal.
- **"Matches_Post"**: funciones "custom", opcionales, codificadas en los canales para procesar los resultados de la descarga de una página cuando las funciones estándar de AH no pueden resolverlos. AH puede tener una serie de **"Profiles"** para procesar por defecto los comportamientos comunes de los canales. Ahora solo está generado el **Profile "default",** pero se podrán añadir más según se vea que hay funciones comunes a grupos de canales.
- **"Matches"**: es el diccionario que resulta del procesado de las funciones "Matches_Post" de los canales o de los "Profiles" de AH. En esencia es la **"Piedra Rosetta"** que convierte las etiquetas que se descargan desde la web en formato BS, Json, etc., a un diccionario con etiquetas comunes para todos los canales y situaciones, que es procesado posteriormente por AH para crear el **Itemlist**.

Esquema de AH

- class AlfaChannelHelper:
  - Variables "self" (ver listado en anexo)
  - Funciones "self" comunes (ver listado en anexo)
    - def create_soup
    - ...
    - def parse_finds_dict
  - sub-class DictionaryAllChannel: canales generalistas
    - list_all: listado
    - section: sub-menus: categorías, años, calidades,...
    - seasons: temporadas de series
    - episodes: episodios temporada de series
    - get_video_options: Findvideos y parte de Play
  - sub-class DictionaryAdultChannel: canales para Adultos
    - list_all: listado
    - section: sub-menus: categorías, canales, pornstars,...
    - get_video_options: Findvideos y parte de Play

Todas las variables "self" y las funciones pueden ser llamadas desde el canal, ej.: AlfaHelper.soup. Es muy **recomendable el uso de estas funciones** en vez de las llamadas a las funciones nativas, porque evitan imports innecesarios, y además permiten corregir centralizadamente cualquier cambio en la función nativa.

Esquema del Canal

El esquema de los canales de adultos es una versión reducida del de los canales generalista:

Canales Generalistas

- from lib.AlfaChannelHelper import DictionaryAllChannel (canales generalistas)
- canonical = {...}
- finds = {...}: metadatos para AH
- AlfaChannel = DictionaryAllChannel(..., finds, ...): generación del objeto AH
- Funciones:
  - mainlist:
    - Autoplay (opcional)
    - Configuración (opcional)
    - Filtertools (opcional)
  - configuracion (opcional)
  - submenu (opcional)
  - section: categorías, años, calidades,... (opcional)
    - section_matches (opcional, hay "profile=default")
  - list_all
    - list_all_matches
  - Si hay series:
    - seasons
      - » seasons_matches (opcional, hay "profile=default")
    - episodios
    - episodesxseason
      - » episodesxseason_matches
  - findvideos
    - findvideos_matches (opcional, hay "profile=default")
  - play (opcional)
    - AlfaChannel.create_soup(...)
  - actualizar_titulos: actualiza con TMDB títulos ambiguos
  - search
  - newest (opcional)

Canales Adultos

- from lib.AlfaChannelHelper import DictionaryAdultsChannel (canales adultos)
- canonical = {...}
- finds = {...}: metadatos para AH
- AlfaChannel = DictionaryAdultsChannel(..., finds, ...): generación del objeto AH
- Funciones:
  - mainlist:
  - submenu (opcional)
  - section: categorías, canales, pornstars,...(opcional)
    - section_matches (opcional, hay "profile=default")
  - list_all
    - list_all_matches
  - findvideos (no se usa, pero es requerido por los tests)
    - findvideos_matches (opcional, hay "profile=default")
  - play (opcional)
    - AlfaChannel.create_soup(...)
  - search

Esquema del diccionario "finds"

Este es un ejemplo del Canal Dontorrent. Es de los más complejos, pero nos servirá de referencia para analizar sus componentes. El contenido de las claves del diccionario tiene un formato orientado a BS. La función de AH que hace de "**parser**" de estas claves es "**parse_finds_dict**":

```
finds = {'find': {'find_all': [{'tag': ['div'], 'class': ['text-center']}]},
         'sub_menu': dict([('find', [{'tag': ['div'], 'class': ['torrents-list']}]),
                           ('find_all', [{'tag': ['a']}])]),
         'categories': {},
         'search': {},
         'get_language': {},
         'get_language_rgx': '',
         'get_quality': {},
         'get_quality_rgx': [],
         'next_page': {},
         'next_page_rgx': [['\/page\/\d+', '/page/%s'], ['&pagina=\d+', '&pagina=%s']],
         'last_page': dict([('find', [{'tag': ['ul'], 'class': ['pagination']}]),
                            ('find_all', [{'tag': ['a'], '@POS': [-2]}]),
                            ('get_text', [{'tag': '', '@STRIP': True, '@TEXT': '(\d+)'}])]),
         'year': {},
         'season_episode': {},
         'seasons': {},
         'season_num': dict([('find', [{'tag': ['a']}]),
                             ('get_text', [{'tag': '', '@STRIP': True, '@TEXT': '(\d+)'}])]),
         'seasons_search_num_rgx': [['(?i)-(\d+)-(?:Temporada|Miniserie)', None], ['(?i)(?:Temporada|Miniserie)-(\d+)', None]],
         'seasons_search_qty_rgx': [['(?i)(?:Temporada|Miniserie)(?:-(.*?)(?:\.|\/|-$|$))', None]],
         'episode_url': '',
         'episodes': dict([('find', [{'tag': ['div'], 'class': ['card shadow-sm p-4']}]),
                           ('find_all', [{'tag': ['tr']}])]),
         'episode_num': [],
         'episode_clean': [],
         'plot': {},
         'findvideos': {'find_all': [{'tag': ['div'], 'class': ['card shadow-sm p-4']}]},
         'title_clean': [['(?i)TV|Online|(4k-hdr)|(fullbluray)|4k| - 4k|(3d)|miniserie|\s*imax|documental|completo', '']],
         'quality_clean': [['(?i)proper|unrated|directors|cut|repack|internal|real|extended|masted|docu|super|duper|amzn|uncensored|hulu', '']],
         'language_clean': [],
         'url_replace': [],
         'controls': {'min_temp': min_temp, 'url_base64': True, 'add_video_to_videolibrary': True, 'cnt_tot': 15,
                      'get_lang': False, 'reverse': False, 'videolab_status': True, 'tmdb_extended_info': True, 'seasons_search': True,
                      'host_torrent': host_torrent, 'btdigg': True, 'duplicates': [], 'dup_list': 'title',
                      'force_find_last_page': [5, 999, 'url'], 'btdigg_quality_control': True},
         'timeout': timeout}
```

Si es necesario el número de etiquetas puede crecer, pero siempre manteniendo los nombres de las claves ya existentes. Veamos las claves más relevantes:

- "**find**": usada por la función "list_all"
- "**sub_menu**": usada por la función de mismo nombre. Se usa cuando en esta función se descarga una página (poco habitual). Lo interesante de este ejemplo es su formato, que es muy habitual.
    - El valor: dict([('find', [{'tag': ['div'], 'class': ['torrents-list']}]), ('find_all', [{'tag': ['a']}])])"
    - Se convierte en: soup.find('div', attr={'class': ['torrents-list']}).find_all('a')
- "**last_page**": paginador: si la web suministra el nº de la última página, éste se pasa en el pie de página:
    - El valor: "dict([('find', [{'tag': ['ul'], 'class': ['pagination']}]),
        - ('find_all', [{'tag': ['a'], '@POS': [-2]}]),
        - ('get_text', [{'tag': '', '@STRIP': True, '@TEXT': '(\d+)'}])])"
    - Se convierte en:
        - matches = soup.find('ul', attr={'class': ['pagination']}).find_all('a')
        - last_page = matches[-2]
        - last_page _text= last_page .get_text(stripe=True)
        - last_page_num = scrapertools.find_single_match(last_page _text, '(\d+)')
- "**next_page_rgx**": sabiendo la última página, el paginador automáticamente va incrementando el nº de página siguiente en la url_next_page. En esta clave se ponen la(s) expresión(es) regular(es) que permiten a AH (con "re.sub") identificar el nº de página actual y reemplazarlo por el de la página siguiente. En este ejemplo hay dos formatos de paginación
- "**seasons_XXX**": en esta web no existe la arquitectura de Serie-temporadas-episodios, sino Temporadas-episodios. Se ha desarrollado una función en Generictools que permite identificar todas las temporadas de una serie y crear en Alfa la arquitectura de Serie-temporadas-episodios compatible con la videoteca. Las claves "**seasons_search_XXX**" son regex que permiten identificar en las urls encontradas las diferentes temporadas. Si existe la arquitectura de Serie-temporadas-episodios hay

que rellenar la clave "**seasons**" con la expresión BS que corresponda para obtener todas las temporadas de una serie

- "**season_episode**": expresión BS para episodio sueltos en "list_all"
- "**episodes**": expresión BS para identificar los episodios dentro de una temporada
- "**findvideos**": expresión BS para identificar los enlaces de un vídeo
- "**title_clean**" y "**quality_clean**": expresiones regex que sirven a AH limpiar los títulos y calidades de algunas web que añaden información incompatible con TMDB. Hay que hacer notar que el formato ['abc', 'xyz'] implica usar "re.sub", mientras que el formato ['abc', None] implica el uso de scrapertools.find_single_match
- "**controls**": es una clave que almacena indicadores importantes que desde el canal permiten dirigir el comportamiento de AH. Por ejemplo,:
    - **'cnt_tot'**: 15 indican a AH que la longitud de la página en Kodi va a ser de 15 líneas, independientemente del tamaño de "matches" que suministre la web. Si el nº de matches recibidas es superior, se almacena el resto en item.matches para pasadas sucesivas. Por el contrario, si el nº es inferior se leen automáticamente páginas sucesivas hasta rellenar la lista. AH tiene un control automático por tiempo máximo (clave 'inicio', 5 segs) para evitar bucles inesperados.
    - **'url_base64'**: True permite descodificar automáticamente urls que vienen ofuscadas o con "acorta-links"
    - **'videolab_status'**: True permite a AH mostrar en las funciones estándar de Alfa el estado de visto/no_visto de series y películas en la videoteca
- "**Timeout**": timeout de la página a descargar

Las funciones BS que soporta "parse_finds_dict" son las siguientes:

- find, find_all
- find_next_sibling, find_next_siblings
- find_next, find_all_next
- select_one, select
- find_parent, find_parents
- find_previous_sibling, find_previous_siblings
- find_previous, find_all_previous
- get_text (alternative a ".text")

Los atributos u opciones adicionales soportados añadidos a las funciones BS son los siguientes:-

- "tagOR": función OR entre funciones BS
- "string": atributo de búsqueda por string
- "@TEXT": buscar con "scrapertools.find_single_match" en el resultado de un función BS
- "@SUB": sustituir con "re.sub" en el resultado de un función BS
- "@ARG": keyword para obtener parte del resultado de un 'find_xxx'. Ej.: 'href'
- "@STRIP": opción para función BS "get_text"
- "@JSON": indica que el resultado de una función BS es un Json y hay que tratarlo como tal. Este atributo puede aportar el "mapa" de conversión de las claves del .json que se descarga, con las claves del .json "matches" con el que opera AH. Ej.; [['video'|'href']', '['url']']]
- "@POS": convierte en soup el nº de posición en la lista resultante de un función BS del tipo "xxx_all"

Este es un ejemplo reducido de diccionario “finds” en un canal tipo para Adultos:

```
finds = {'find': dict([('find', [{'tagOR': ['ul'], 'id': ['mostRecentVideosSection']},
                                 {'tagOR': ['ul'], 'class': ['row-5-thumbs']},
                                 {'tag': ['li'], 'class': ['sniperModeEngaged']}]), ('find_parent', [{'tag': []}]),
                       ('find_all', [{'tag': ['li'], 'class': ['pcVideoListItem']}])]),
        'categories': {'find_all': [{'tag': ['div'], 'class': ['category-wrapper', 'channelsWrapper']}]},
        'search': {},
        'get_quality': {},
        'get_quality_rgx': '',
        'next_page': dict([('find', [{'tag': ['div'], 'class': ['paginationGated']}]),
                           ('find_all', [{'tag': ['a'], '@POS': [-1], '@ARG': 'href'}])]),
        'next_page_rgx': [['\/page\/\d+', '/page/%s']],
        'last_page': {},
        'plot': {},
        'findvideos': {},
        'title_clean': [['[\(|\[]\s*[\)|\]]', ''], ['(?i)\s*videos*\s*', '']],
        'quality_clean': [['(?i)proper|unrated|directors|cut|repack|internal|real|extended|masted|docu|super', '']],
        'url_replace': [],
        'controls': {'url_base64': False, 'cnt_tot': 30, 'reverse': False},
        'timeout': timeout}
```

Conviene resaltar el uso de algunos atributos y opciones:

- “**find**” utiliza la opción “**tagOR**”. Esto se utiliza porque las diferentes páginas de “list_all” que usan esta función BS tienen diferentes formatos:
    - El valor: dict([('find', [{'tagOR': ['ul'], 'id': ['mostRecentVideosSection']},
        - {'tagOR': ['ul'], 'class': ['row-5-thumbs']},
        - {'tag': ['li'], 'class': ['sniperModeEngaged']}]), ('find_parent', [{'tag': []}]),
        - ('find_all', [{'tag': ['li'], 'class': ['pcVideoListItem']}])])
    - Se convierte en:
        - matches = soup.find(‘ul’, attr={'id': ['mostRecentVideosSection']}) or soup.find(‘ul’, attr={'class': [''row-5-thumbs'']})
        - If matches:
            - matches = soup.find_all(‘li’, attr={'class': [' pcVideoListItem ']})
        - else:
            - matches = soup.find(‘li’, attr={'class': [' sniperModeEngaged ']}).find_parent().find_all(‘li’, attr={'class': [' pcVideoListItem ']})
- “**categories**” tiene **dos argumentos** dentro de ‘class’, funcionado como OR
- “**next_page**” se usa en lugar de “last_page”. En este caso retorna la url de la siguiente página

NOTAS sobre "finds":

OrderedDict:

Habrás notado que algunas claves de "**finds**" tienen el valor en formato "**dict**", mientras otras no.

El problema viene de la compatibilidad con Kodi 18 y anteriores, más específicamente por la compatibilidad con **Python 2.7**. Hasta la versión 3.6 de Python **los diccionarios no garantizaban que el orden de las claves fuera el mismo que en el que se habían introducido**.

El diseño de "finds" exige que las funciones BS especificadas se ejecuten en ese orden. Por ejemplo la clave descrita más arriba:

- "dict([('find', [{'tag': ['ul'], 'class': ['pagination']}]),
- ('find_all', [{'tag': ['a'], '@POS': [-2]}]),
- ('get_text', [{'tag': '', '@STRIP': True, '@TEXT': '(\d+)'}])])"

En Python 2.7 se ejecutaría en este orden:

- find('ul', class…).get_text(…).find_all(…)

Obviamente la función "find_all(…)" daría error porque el resultado de get_text(…) es un texto sin formato BS.

La forma de resolverlo en Python 2.7 es usar "**from collections import OrderedDict** as dict". De esta forma se suplanta en Python 2.7 la función nativa "dict" por "OrderedDict". Como OrderedDict tiene un impacto en rendimiento y en consumo de memoria, he ampliado el import a "**if not PY3:** _dict = dict; from collections import OrderedDict as dict". De esta forma sólo usa OrderedDict en Python 2.7 (y solo en la claves que usan "dict") mientras en Python3 usa la función nativa, rápida y ordenada.

Esto ocasionaba un problema colateral: las preguntas "if isinstance(soup, **dict**)" dejan de funcionar porque ahora "dict" está suplantado por "OrderedDict". La alternativa es salvar la función "dict" nativa como "**_dict**" antes de ser mutada, para luego usar "if isinstance(soup, **_dict**)". De ahí la sentencia final del import ""if not PY3: **_dict = dict**; from …". Por último, para que funcione transparentemente en Python 3, modificamos la sentencia inicial "if sys.version_info[0] >= 3: PY3 = True; unicode = str; unichr = chr; long = int; **_dict = dict**".

Creo que esta solución es la menos invasiva y garantiza la compatibilidad entre Python 2.7 y Python 3.6+. Afortunadamente el formato del "constructor" de "OrderedDict" y de "dict" es el mismo.

"OrderedDict" sólo se debe usar para aquellos diccionarios donde se requiera un orden. Para el resto se seguirá utilizado el típico "pepe = {}"

Debugging:

Para obtener la mayor productividad como desarrollador, es muy conveniente activar en los **settings.xml** de Alfa (además del "debug"), el siguiente parámetro. No está disponible en el menú de Ajustes y ahora solo se activa temporalmente mientras se genera un informe de error:

<setting id="debug_report" default="true">true</setting>

Con este parámetro activado **el log se enriquece enormemente**, dando información extensiva de los pasos dentro de AH, sobre todo en el **"parseo" de "finds"**. También da información clave sobre los **parámetros pasados en las descargas de las webs**. Esto reduce en gran medida el tiempo necesario para analizar los problemas de un canal.

Estructura de funciones “matches_post” de canal para Adultos

Vamos a analizar un par de funciones “matches_post” de los canales PornHub y PornTube:

```python
def list_all(item):
    logger.info()

    return AlfaChannel.list_all(item, matches_post=list_all_matches, **kwargs)


def list_all_matches(item, matches_int, **AHkwargs):
    logger.info()

    matches = []
    findS = AHkwargs.get('finds', finds)
    item.c_type = 'peliculas'

    for elem in matches_int:
        elem_json = {}
        #logger.error(elem)

        try:
            if elem.find('div', class_='thumbTextPlaceholder'): continue
            elem_json['url'] = elem.a.get('href', '')
            elem_json['title'] = elem.a.get('title', '')
            elem_json['thumbnail'] = elem.img.get('src', '')
            elem_json['canal'] = elem.find('div', class_='usernameWrap').get_text(strip=True) \
                                 if elem.find('div', class_='usernameWrap') else ''
            elem_json['stime'] = elem.find('var', class_='duration').get_text(strip=True)
            elem_json['quality'] = elem.find('span', class_='hd-thumbnail').get_text(strip=True) if elem.find('span', class_='hd-thumbnail') else ''
            elem_json['premium'] = elem.find('i', class_='premiumIcon') or ''
            if elem.find('div', class_='videoDetailsBlock') and elem.find('div', class_='videoDetailsBlock').find('span', class_='views'):
                elem_json['views'] = elem.find('div', class_='videoDetailsBlock').find('span', class_='views')\
                                     .get_text('|', strip=True).split('|')[0]
            #elem_json['action'] = 'findvideos'

        except:
            logger.error(elem)
            logger.error(traceback.format_exc())
            continue

        if elem_json['premium']: continue
        if not elem_json['url']: continue

        matches.append(elem_json.copy())

    return matches
```

Es una función muy representativa de este tipo de canales. De hecho en este caso no haría falta porque forma parte del “profile=’default’” de la función “list_all” de AH. La dejamos ahí como ejemplo de utilización

Elementos a tener en cuenta:

- Las llamadas “matches_post” siempre tienen los mismos parámetros. Se usa la capacidad de AHkwargs para pasar parámetros específicos, típicamente: **“soup”, “finds”, “function” y “matches”** (cuando están disponibles)
- “findS”: siempre hay que recargar esta variable local con el “finds” que viene en AHkwargs. Ha podido ser actualizado con respecto al “finds” que se define al principio del canal
- “for”: las acciones a realizar con “matches_int” se deben poner dentro de un “try”, para evitar que el fallo de un match no estropee el resultado final
- “elem_json”: es el formato estándar que espera la función correspondiente de AH. La gran mayoría de las claves son opcionales, salvo “url”, etc…
- “elem_json[‘action’]” por defecto está en “play”, en la llamada inicial a AH: movie_action='play', pero se puede cambiar a voluntad
- En esta zona no hace falta limpiar títulos ni calidades (opciones “finds[‘title_clean’] y “finds[‘quality_clean’] permite a AH hacerlo después).
- El formato final del título en canales para Adultos se realiza por AH con la función “unify_custom”, una versión super reducida del “unify” de Alfa. El valor de “elem_json[‘stime’]” se formatea si es necesario en “AH.convert_time”

Otro formato en principio distinto, como es el de respuestas en la web en formato .json, al final se resuelve de forma muy similar:

```python
def list_all(item):
    logger.info()
    kwargs['soup'] = False
    kwargs['json'] = True

    return AlfaChannel.list_all(item, matches_post=list_all_matches, **kwargs)

def list_all_matches(item, matches_int, **AHkwargs):
    logger.info()

    matches = []
    findS = AHkwargs.get('finds', finds)
    item.c_type = 'peliculas'

    if matches_int.get('embedded', {}).get('videos', {}): matches_int = matches_int['embedded'].copy()
    matches_int = matches_int.get('videos', {}).copy()
    if not matches_int: return matches

    if matches_int.get('pages'): AlfaChannel.last_page = matches_int.get('pages', 0)
    if matches_int.get('limit'): AlfaChannel.finds['controls']['cnt_tot'] = matches_int.get('limit', findS['controls']['cnt_tot'])

    matches_int = matches_int.get("_embedded", {}).get("items", {})
    if not matches_int: return matches

    for elem in matches_int:
        elem_json = {}
        #logger.error(elem)

        try:
            elem_json['url'] = "%sembed/%s" % (host, elem.get('uuid', 0))
            elem_json['title'] = elem.get('title', '')
            elem_json['thumbnail'] = elem.get('thumbnailsList', [])[0] if elem.get('thumbnailsList', []) else ''
            elem_json['stime'] = elem.get('durationInSeconds', 0)
            elem_json['quality'] = 'HD' if elem.get('isHD') else ''

        except:
            logger.error(elem)
            logger.error(traceback.format_exc())
            continue

        if not elem_json['url']: continue

        matches.append(elem_json.copy())
    return matches
```

Hay que asegurarse que antes de llamar a AH se pone en kwargs los parámetros “**soup=False**” y “**json=True**”. Si se van a usar de forma general para todo el canal, se pueden poner al principio del código del canal

Ya en la función “matches_post”, el objetivo es modificar “matches_int” de tal forma que apunte a la lista de elementos-diccionarios con la información necesaria para crear un “elem_json” estándar. En realidad es una conversión de claves del diccionario.

Nota adicionales

- Siempre que se quiera obtener información de un objeto “soup” o de un “dict”, se debe usar el formato “**elem.get(‘keyword’, valor-por-defecto).** Así se evita que la ausencia de una “keyword” fuerce un error de programación. De cara al usuario queda mejora una pantalla con “no hay elementos” que un error.

- En la función “**search**” se debe mantener al menos este código, algunos campos reservados para futuro uso:

```
def search(item, texto, **AHkwargs):
    logger.info()
    kwargs.update(AHkwargs)

    item.url = "%s?XXX=%s" % (host, texto.replace(" ", "+"))

    try:
        if texto:
            item.c_type = "search"
            item.texto = texto
            return list_all(item)
        else:
            return []

    # Se captura la excepción, para no interrumpir al buscador global si un canal falla
    except:
        for line in sys.exc_info():
            logger.error("%s" % line)
        return []
```

- La función “**play**” no es obligatorio ponerla, a no ser que realice una función adicional a la estándar de Alfa.

En **platformtools** hay este código que creo satisface a los canales de adultos en la mayoría de las situaciones:

```
    if not item.server:
        itemlist = servertools.get_servers_itemlist([item])
        if itemlist:
            item = itemlist[0]
        else:
            item.server = "directo"
```

- **Estandarización de Estilos**

En aras a conseguir un formato en los canales entendible por cualquier programador, es muy recomendable conservar el mismo estilo en todos los canales. Es recomendable usar un **canal modelo** y usarlo como **plantilla** a la hora de convertir los canales existentes. Por ejemplo, la parte inicial del canal debería ser la misma, conservado las mismas variables aunque no se usen en ese canal, y luego las funciones en el mismo orden:

```
# -*- coding: utf-8 -*-
# -*- Channel PornTube -*-
# -*- Created for Alfa-addon -*-
# -*- By the Alfa Develop Group -*-

import sys
PY3 = False
if sys.version_info[0] >= 3: PY3 = True; unicode = str; unichr = chr; long = int; _dict = dict

import re
import traceback
if not PY3: _dict = dict; from collections import OrderedDict as dict

from core.item import Item
from core import servertools
from core import scrapertools
from core import jsontools
from channelselector import get_thumb
from platformcode import config, logger
from channels import filtertools, autoplay
from lib.AlfaChannelHelper import DictionaryAdultChannel

IDIOMAS = {}
list_language = list(set(IDIOMAS.values()))
list_quality = []
list_quality_movies = []
list_quality_tvshow = []
list_servers = []
forced_proxy_opt = 'ProxySSL'

canonical = {...}
host = canonical['host'] or canonical['host_alt'][0]
timeout = 5
kwargs = {}
debug = config.get_setting('debug_report', default=False)
movie_path = ''
tv_path = ''
language = []
url_replace = []

finds = {...}
AlfaChannel = DictionaryAdultChannel(host, movie_path=movie_path, tv_path=tv_path, movie_action='play', canonical=canonical, finds=finds,
                                     idiomas=IDIOMAS, language=language, list_language=list_language, list_servers=list_servers,
                                     list_quality_movies=list_quality_movies, list_quality_tvshow=list_quality_tvshow,
                                     channel=canonical['channel'], actualizar_titulos=True, url_replace=url_replace, debug=debug)
```

ANEXO

Variables SELF

```
def __init__(self, host, timeout=15, channel='', movie_path="/movies", tv_path="/serie",
             movie_action="findvideos", tv_action="seasons", forced_proxy_opt='',
             list_language=[], list_quality=[], list_quality_movies=[], list_quality_tvshow=[],
             list_servers=[], language=[], idiomas={}, IDIOMAS_TMDB={0: 'es', 1: 'en', 2: 'es,en'},
             actualizar_titulos=True, canonical={}, finds={}, debug=False, url_replace=[]):
```

self.url
self.host = host
self.host_torrent = finds.get('controls', {}).get('host_torrent', self.host)
self.timeout = timeout
self.doo_url = "%swp-admin/admin-ajax.php" % host
self.channel = channel or canonical.get('channel', '')
self.movie_path = movie_path
self.tv_path = tv_path
self.movie_action = movie_action
self.tv_action = tv_action
self.forced_proxy_opt = forced_proxy_opt
self.list_language = list_language
self.list_quality = self.list_quality_tvshow = list_quality or list_quality_tvshow
self.list_quality_movies = list_quality_movies
self.list_servers = list_servers
self.language = language
self.idiomas = idiomas or IDIOMAS
self.IDIOMAS_TMDB = IDIOMAS_TMDB
self.actualizar_titulos = actualizar_titulos
self.canonical = canonical
self.finds = finds
self.profile = self.finds.get('controls', {}).get('profile', DEFAULT)
self.url_replace = url_replace
self.Window_IsMedia = True
self.season_colapse = True
self.unescape = False
self.btdigg = False
self.btdigg_search = False
self.last_page = ''
self.itemlist = []
self.response = httptools.build_response(HTTPResponse=True): crea estructura vacía de response
self.response_proxy = self.response.proxy__: IP del proxy o ProxyWeb si se ha usado en la descarga
self.response_preferred_proxy_ip = '': IP de ProxySSL reusable, usada en la última descarga de un canal
self.alfa_domain_web_list = Lista de PoxySSL verificados
self.headers = {}
self.SUCCESS_CODES = httptools.SUCCESS_CODES
self.REDIRECTION_CODES = httptools.REDIRECTION_CODES
self.PROXY_CODES = httptools.PROXY_CODES
self.NOT_FOUND_CODES = httptools.NOT_FOUND_CODES

```
self.CLOUDFLARE_CODES = httptools.CLOUDFLARE_CODES
self.OPENSSL_VERSION = httptools.OPENSSL_VERSION
self.ssl_version = httptools.ssl_version
self.ssl_context = httptools.ssl_context
self.patron_local_torrent
self.TEST_ON_AIR = httptools.TEST_ON_AIR
self.ASSISTANT_VERSION = config.get_setting('assistant_version')
```

Funciones comunes de AH disponibles desde los canales:

```
def create_soup(self, url, **kwargs):
def define_content_type(self, new_item, contentType=''):
def add_video_to_videolibrary(self, item, itemlist, contentType='tvshow'):
def do_url_replace(self, url, url_replace=[]):
def do_quote(self, url, plus=True):
def do_unquote(self, url):
def do_urlencode(self, post):
def urljoin(self, domain, url):
def obtain_domain(self, url, sub=False, point=False, scheme=False):
def channel_proxy_list(self, url, forced_proxy=None):
def get_cookie(self, url, name, follow_redirects=False):
def do_soup(self, data, encoding='utf-8'):
def do_actualizar_titulos(self, item):
def do_seasons_search(self, item, matches, **AHkwargs):
def convert_url_base64(self, url, host='', referer=None, rep_blanks=True, force_host=False):
def is_local_torrent(self, url):
def find_btdigg_list_all(self, item, matches, channel_alt, **AHkwargs):
def find_btdigg_seasons(self, item, matches, domain_alt, **AHkwargs):
def find_btdigg_episodes(self, item, matches, domain_alt, **AHkwargs):
def find_btdigg_findvideos(self, item, matches, domain_alt, **AHkwargs):
def check_filter(self, item, itemlist, **AHkwargs):
def unify_custom(self, title, item, elem, **AHkwargs):
def get_language_and_set_filter(self, elem, elem_json):
def find_quality(self, elem_in, item):
def find_language(self, elem_in, item):
def convert_size(self, size):
def convert_time(self, seconds):
def manage_torrents(self, item, elem, lang, soup={}, finds={}, **kwargs):
def parse_finds_dict(self, soup, finds, year=False, next_page=False, c_type=''):
```

class DictionaryAllChannel(AlfaChannelHelper):

```
def list_all(self, item, data='', matches_post=None, postprocess=None, generictools=False, finds={}, **kwargs):
def section(self, item, data= '', action="list_all", matches_post=None, postprocess=None,
        section_list={}, finds={}, **kwargs):
def seasons(self, item, data='', action="episodesxseason", matches_post=None, postprocess=None,
        seasons_search_post=None, generictools=True, seasons_list={}, finds={}, **kwargs):
def episodes(self, item, data='', action="findvideos", matches_post=None, postprocess=None,
        generictools=False, episodes_list={}, finds={}, **kwargs):
def get_video_options(self, item, url, data='', langs=[], matches_post=None, postprocess=None,
            verify_links=False, generictools=False, findvideos_proc=False, finds={}, **kwargs):
```

class DictionaryAdultChannel(AlfaChannelHelper):

```
def list_all(self, item, data='', matches_post=None, postprocess=None, generictools=False, finds={}, **kwargs):
def section(self, item, data= '', action="list_all", matches_post=None, postprocess=None,
        section_list={}, finds={}, **kwargs):
def get_video_options(self, item, url, data='', langs=[], matches_post=None, postprocess=None,
            verify_links=False, generictools=False, findvideos_proc=False, finds={}, **kwargs):
```