

Estructura y funcionamiento de AlfaChannelHelper (AH)

Este módulo está 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 las expresiones regulares (en formato BeautifulSoup4 (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”** en algunas funciones, pero se podrán añadir más según se vea que hay funciones comunes a grupos de canales.
- **“Matches”**: es el diccionario que resulta del procesamiento 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.response**. 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 import AlfaChannelHelper`
- `from AlfaChannelHelper import DictionaryAllChannel` (canales generalistas)
- `from AlfaChannelHelper import` (módulos necesarios en el canal ya importados por `AlfaChannelHelper`)
- Listas de idiomas, servidores y calidades (por defecto desde `AlfaChannelHelper`)
- `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 import AlfaChannelHelper`
- `from AlfaChannelHelper import DictionaryAdultsChannel` (canales adultos)
- `from AlfaChannelHelper import` (módulos necesarios en el canal ya importados por `AlfaChannelHelper`)
- Listas de idiomas, servidores y calidades (por defecto desde `AlfaChannelHelper`)
- `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”:

```
80 finds = {'find': {'find_all': [{'tag': ['div'], 'class': ['text-center']}]},
81 'sub_menu': dict(['find', [{'tag': ['div'], 'class': ['torrents-list']}]},
82 ('find_all', [{'tag': ['a']}])),
83 'categories': {},
84 'search': {},
85 'get_language': {},
86 'get_language_rgx': '',
87 'get_quality': {},
88 'get_quality_rgx': [],
89 'next_page': {},
90 'next_page_rgx': [['\page/\d+', '/page/%s'], ['&pagina=\d+', '&pagina=%s']],
91 'last_page': dict(['find', [{'tag': ['ul'], 'class': ['pagination']}]},
92 ('find_all', [{'tag': ['a'], '@POS': [-2]}]),
93 ('get_text', [{'tag': '', '@STRIP': True, '@TEXT': '(\d+)'}])),
94 'year': {},
95 'season_episode': {},
96 'seasons': {},
97 'season_num': dict(['find', [{'tag': ['a']}]},
98 ('get_text', [{'tag': '', '@STRIP': True, '@TEXT': '(\d+)'}])),
99 'seasons_search_num_rgx': [['(?(i)-(\d+)-(?:Temporada|Miniserie)', None], ['(?(i)?:Temporada|Miniserie)-(\d+)', None]],
100 'seasons_search_qty_rgx': [['(?(i)?:Temporada|Miniserie)?:-(.*)?:\./|-$|$', None]],
101 'episode_url': '',
102 'episodes': dict(['find', [{'tag': ['div'], 'class': ['card shadow-sm p-4']}]},
103 ('find_all', [{'tag': ['tr']}])),
104 'episode_num': [],
105 'episode_clean': [],
106 'plot': {},
107 'findvideos': {'find_all': [{'tag': ['div'], 'class': ['card shadow-sm p-4']}]},
108 'title_clean': [['(?(i)TV|Online|(4k-hdr)|(fullblurray)|4k| - 4k|(3d)|miniserie|s*imax|documental|completo', '''],
109 'quality_clean': [['(?(i)proper|unrated|directors|cut|repack|internal|real|extended|masted|docu|super|duper|amzn|uncensored|hulu', '''],
110 'language_clean': [],
111 'url_replace': [],
112 'controls': {'min_temp': min_temp, 'url_base64': True, 'add_video_to_videolibrary': True, 'cnt_tot': 15,
113 'get_lang': False, 'reverse': False, 'videolab_status': True, 'tmdb_extended_info': True, 'seasons_search': True,
114 'host_torrent': host_torrent, 'btidgg': True, 'duplicates': [], 'dup_list': 'title',
115 'force_find_last_page': [5, 999, 'url'], 'btidgg_quality_control': True},
116 '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.
 - o El valor: dict(['find', [{'tag': ['div'], 'class': ['torrents-list']}]}, ('find_all', [{'tag': ['a']}])))”
 - o 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:
 - o El valor: “dict(['find', [{'tag': ['ul'], 'class': ['pagination']}]}, ('find_all', [{'tag': ['a'], '@POS': [-2]}]), ('get_text', [{'tag': '', '@STRIP': True, '@TEXT': '(\d+)'}]))”
 - o Se convierte en:
 - matches = soup.find('ul', attr={'class': ['pagination']}).find_all('a')
 - last_page = matches[-2]
 - last_page_text= last_page.get_text(strip=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 (ver lista completa en Anexo). Por ejemplo,:
 - o **‘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.
 - o **‘url_base64’**: True permite decodificar automáticamente urls que vienen ofuscadas o con **“acorta-links”**
 - o **‘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 las páginas a descargar

Las funciones BS que soporta “parse_finds_dict” son las siguientes:

- | | |
|---|---|
| - find, find_all | - find_parent, find_parents |
| - find_next_sibling, find_next_siblings | - find_previous_sibling, find_previous_siblings |
| - find_next, find_all_next | - find_previous, find_all_previous |
| - select_one, select | - 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 una función BS
- **“@EVAL”**: ejecuta **ast.literal_eval(str(match))** sobre el resultado de una función BS
- **“@SUB”**: sustituir con **“re.sub”** en el resultado de una 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.: **‘@JSON’**: **‘description|plot’, ‘contentUrl|url’**. También sirve para identificar una lista de jsons dentro del json descargado, y poder pasarlo a **“matches_int”**. Ej.: **‘@JSON’**: **‘props,pageProps,post,seasons|DEFAULT’**, o **‘@JSON’**: **‘DEFAULT’** para que pase todo el json.
- **“@POS”**: convierte en soup el nº de posición en la lista resultante de una función BS del tipo **“xxx_all”**

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

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

Conviene resaltar el uso de algunos atributos y opciones:

- “**find**” utiliza la opción “**tagOR**”. Esto se utiliza porque las diferentes páginas web en “list_all” que usan esta función BS tienen diferentes formatos:
 - o 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']}]])
 - o 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’, funcionando 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:

```
117 def list_all(item):
118     logger.info()
119
120     return AlfaChannel.list_all(item, matches_post=list_all_matches, **kwargs)
121
122
123 def list_all_matches(item, matches_int, **AHkwargs):
124     logger.info()
125
126     matches = []
127     finds = AHkwargs.get('finds', finds)
128     item.c_type = 'peliculas'
129
130     for elem in matches_int:
131         elem_json = {}
132         #logger.error(elem)
133
134         try:
135             if elem.find('div', class_='thumbTextPlaceholder'): continue
136             elem_json['url'] = elem.a.get('href', '')
137             elem_json['title'] = elem.a.get('title', '')
138             elem_json['thumbnail'] = elem.img.get('src', '')
139             elem_json['canal'] = elem.find('div', class_='usernameWrap').get_text(strip=True) \
140                 if elem.find('div', class_='usernameWrap') else ''
141             elem_json['stime'] = elem.find('var', class_='duration').get_text(strip=True)
142             elem_json['quality'] = elem.find('span', class_='hd-thumbnail').get_text(strip=True) if elem.find('span', class_='hd-thumbnail') else ''
143             elem_json['premium'] = elem.find('i', class_='premiumIcon') or ''
144             if elem.find('div', class_='videoDetailsBlock') and elem.find('div', class_='videoDetailsBlock').find('span', class_='views'):
145                 elem_json['views'] = elem.find('div', class_='videoDetailsBlock').find('span', class_='views')\
146                     .get_text(' ', strip=True).split('|')[0]
147             elem_json['action'] = 'findvideos'
148
149         except:
150             logger.error(elem)
151             logger.error(traceback.format_exc())
152             continue
153
154         if elem_json['premium']: continue
155         if not elem_json['url']: continue
156
157         matches.append(elem_json.copy())
158
159     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... (ver lista completa en Anexo de etiquetas procesadas por AH)
- Logger.error: durante el desarrollo y test conviene tener esta opción activada. Para producción hay que desactivarla.
- “elem_json[‘action’]” por defecto está en “play” para canales de adultos, 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:

```
120 def list_all(item):
121     logger.info()
122
123     if item.c_type == 'search':
124         kwargs['headers'] = {"x-requested-with": "XMLHttpRequest"}
125         kwargs['json'] = True # Es necesario SOLO si devuelve un json puro, sin estructura html
126         kwargs['soup'] = False # Indica que no se trate como BS4
127
128     return AlfaChannel.list_all(item, matches_post=list_all_matches, **kwargs)
129
130 def list_all_matches(item, matches_int, **kwargs):
131     logger.info()
132
133     matches = []
134     finds = AHkwargs.get('finds', finds)
135     """
136     ----- Define el finds que el resultado de la descarga es un una lista de jsons.
137     | Como no tiene paginación, se ajusta en función de la entradas recibidas
138     | 'search': dict([('find', [{'tag': ['body'],}],),
139     | | | | | ('get_text', [{'tag': '', '@STRIP': False, '@JSON': 'data,m&s|DEFAULT'}])]),
140     | | | | |
141     | """
142     if item.c_type == 'search' and AlfaChannel.last_page == 99999:
143         AlfaChannel.last_page = int((float(len(matches_int)) / float(AlfaChannel.cnt_tot)) + 0.999999)
144
145     for elem in matches_int:
146         elem_json = {}
147         #logger.error(elem)
148
149         try:
150             elem_json['url'] = elem.get('slug', '') or elem.a.get("href", "")
151             elem_json['title'] = elem.get('title', '') or elem.a.p.get_text(strip=True)
152             elem_json['thumbnail'] = elem.get('cover', '') or elem.a.figure.img.get("src", "")
153             elem_json['year'] = elem.get('release_year', '') or elem.a.span.get_text(strip=True)
154             if item.c_type == 'search': elem_json['mediatype'] = 'movie' if movie_path in elem_json['url'] else 'tvshow'
155
156         except:
157             logger.error(elem)
158             logger.error(traceback.format_exc())
159             continue
160
161         if not elem_json['url']: continue
162
163         matches.append(elem_json.copy())
164
165     return matches
```

Hay que asegurarse que antes de llamar a AH se pone en kwargs los parámetros “soup=False” y “json=True” si la respuesta no tiene estructura html. Si se van a usar de forma general para todo el canal, se pueden poner al principio del código del canal.

Se puede observar que el diccionario elem_json se va rellenando indistintamente con etiquetas que vienen de un json o de un bloque BS.

También hay que notar en el comentario de la línea 139 el comando '@JSON': 'data,m&s|DEFAULT'. En ese caso se le está indicando al ‘parser’ que del json recibido, encuentre la etiqueta ‘data’, y que dentro tiene dos etiquetas: ‘m’ y ‘s’. Con el símbolo ‘&’ le indicamos que queremos que en la lista final de jsons incluya las DOS ramas.

Cuando se recibe un json normalmente no hay indicador de paginación y en muchas ocasiones el json incluye la totalidad de la respuesta. En esta situación, con las líneas 141-142 se simula el last_page y se pasa a al AH.

Colores de Itemlist: Unify_custom para canales de Adultos

El método “unify_custom” pone los colores a los títulos, calidades, duración, visitas, canales, stars, etc. El esquema de colores usado está **basado en el perfil de colores seleccionado por el usuario** la el “unify” de Alfa. El razonamiento es que el usuario debería ver el mismo esquema de colores en todos los canales, bien sean generalistas o de adultos. El esquema seleccionado es este:


```

color_setting = self.color_setting
color_default = 'white'
text = color_setting.get('movie', color_default)
stime = color_setting.get('year', color_default)
quality = color_setting.get('quality', color_default)
language = color_setting.get('cast', color_default)
server = color_setting.get('server', color_default)
canal = color_setting.get('tvshow', color_default)
views = color_setting.get('rating_3', color_default)
library = color_setting.get('library', color_default)
page = color_setting.get('update', color_default)
star = color_setting.get('rating_3', color_default)
error = 'red'

```

En la llamada a la función se pasa el nombre de la función llamadora, con lo que se puede ajustar la respuesta.

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 mejor 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:

```

228 def search(item, texto, **AHkwargs):
229     logger.info()
230     kwargs.update(AHkwargs)
231
232     item.url = "%s?XXX=%s" % (host, texto.replace(" ", "+"))
233
234     try:
235         if texto:
236             item.c_type = "search"
237             item.texto = texto
238             return list_all(item)
239         else:
240             return []
241
242     # Se captura la excepción, para no interrumpir al buscador global si un canal falla
243     except:
244         for line in sys.exc_info():
245             logger.error("%s" % line)
246         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:

```

1315 if not item.server:
1316     itemlist = servertools.get_servers_itemlist([item])
1317     if itemlist:
1318         item = itemlist[0]
1319     else:
1320         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, conservando las mismas variables aunque no se usen en ese canal, y luego las funciones en el mismo orden:

```
1  # -*- coding: utf-8 -*-
2  # -*- Channel NAME -*-
3  # -*- Created for Alfa Addon -*-
4  # -*- By the Alfa Development Group -*-
5
6  import sys
7  PY3 = False
8  if sys.version_info[0] >= 3: PY3 = True; unicode = str; unichr = chr; long = int; _dict = dict
9
10 from lib import AlfaChannelHelper
11 if not PY3: _dict = dict; from AlfaChannelHelper import dict
12 from AlfaChannelHelper import DictionaryAllChannel
13 from AlfaChannelHelper import re, traceback, time, base64, xbmcgui
14 from AlfaChannelHelper import Item, servertools, scrapertools, jsontools, get_thumb, config, logger, filtertools, autoplay
15
16 IDIOMAS = AlfaChannelHelper.IDIOMAS # o IDIOMAS_T o IDIOMAS_ANIME o IDIOMAS_A
17 list_language = list(set(IDIOMAS.values()))
18 list_quality_movies = AlfaChannelHelper.LIST_QUALITY_MOVIES # o LIST_QUALITY_MOVIES_T o LIST_QUALITY_MOVIES_A
19 list_quality_tvshow = AlfaChannelHelper.LIST_QUALITY_TVSHOW
20 list_quality = list_quality_movies + list_quality_tvshow
21 list_servers = AlfaChannelHelper.LIST_SERVERS # o LIST_SERVERS_T o LIST_SERVERS_A
22 forced_proxy_opt = 'ProxySSL'
23
24 canonical = {...}
25 host = canonical['host'] or canonical['host_alt'][0]
26
27 timeout = 5
28 kwargs = {}
29 debug = config.get_setting('debug_report', default=False)
30 movie_path = "película"
31 tv_path = 'serie/'
32 language = []
33 url_replace = []
34
35 finds = {...}
36 AlfaChannel = DictionaryAllChannel(host, movie_path=movie_path, tv_path=tv_path, canonical=canonical, finds=finds,
37                                   idiomas=IDIOMAS, language=language, list_language=list_language, list_servers=list_servers,
38                                   list_quality_movies=list_quality_movies, list_quality_tvshow=list_quality_tvshow,
39                                   channel=canonical['channel'], actualizar_titulos=True, url_replace=url_replace, debug=debug)
```

Para las etiquetas “language” y “quality” es importante normalizar el formato. Si no se tiene la certeza de tener el valor adecuado, es recomendable usar este formato para que luego AH lo normalice:

- elem_json['language'] = '%s' % elem.find...
- elem_json['quality'] = '%s' % elem.find...

ANEXO

Import en AH

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

```
import re
import traceback
import time
import base64
import xbmcgui
window = xbmcgui.Window(10000) or None
if not PY3: _dict = dict; from collections import OrderedDict as dict
```

```
from channelselector import get_thumb
from core import scrapertools
from core import servertools
from core import jsontools
from core.item import Item
from platformcode import config
from platformcode import logger
from channels import filtertools
from modules import autoplay
```

Variables SELF en class AlfaChannelHelper:

```
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=[]):

    from core import httptools
    from platformcode import unify
    global DEBUG
    DEBUG = self.DEBUG = debug if not httptools.TEST_ON_AIR else False

    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', "")
    if self.channel and canonical and 'host' in canonical and not canonical['host'] and canonical.get('host_alt',
[]): config.set_setting("current_host", canonical['host_alt'][0], channel=self.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_servers = list_servers or LIST_SERVERS
self.list_quality_movies = list_quality_movies or LIST_QUALITY_MOVIES
self.list_quality_tvshow = list_quality_tvshow or LIST_QUALITY_TVSHOW
self.list_quality = list_quality or self.list_quality_movies + self.list_quality_tvshow
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_collapse = True
self.unescape = False
self.btdigg = False
self.btdigg_search = False
self.last_page = 0
self.curr_page = 0
self.next_page_url = ""
self.itemlist = []
self.color_setting = unify.colors_file[UNIFY_PRESET]
self.window = window
self.Comment = None
self.httptools = httptools
self.response = self.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 = jsontools.load(window.getProperty("alfa_domain_web_list")) \
    if window.getProperty("alfa_domain_web_list") else {} # Lista de ProxySSL verificados
self.headers = {}
self.SUCCESS_CODES = self.httptools.SUCCESS_CODES
self.REDIRECTION_CODES = self.httptools.REDIRECTION_CODES
self.PROXY_CODES = self.httptools.PROXY_CODES
self.NOT_FOUND_CODES = self.httptools.NOT_FOUND_CODES
self.CLOUDFLARE_CODES = self.httptools.CLOUDFLARE_CODES
self.OPENSLL_VERSION = self.httptools.OPENSLL_VERSION
self.ssl_version = self.httptools.ssl_version
self.ssl_context = self.httptools.ssl_context
self.patron_local_torrent = '(?i)(?:(?:\\|\\V)[^\\[]+\\[\\w+\\](?:\\|\\V)[^\\[]+\\[\\w+\\]_\\d+\\.torrent|magnet\\:)'
self.TEST_ON_AIR = self.httptools.TEST_ON_AIR
self.KWARGS = KWARGS.copy()
try:
    self.ASSISTANT_VERSION = config.get_setting('assistant_version', default="").split('.')
    self.ASSISTANT_VERSION = tuple((int(self.ASSISTANT_VERSION[0]), int(self.ASSISTANT_VERSION[1]),
int(self.ASSISTANT_VERSION[2])))
except Exception:
    self.ASSISTANT_VERSION = tuple()

```

Funciones comunes de AH disponibles desde los canales:

```
def create_soup(self, url, **kwargs):
def define_content_type(self, new_item, contentType=""):
def get_color_from_settings(self, label, default='white'):
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, safe="", encoding=None, errors=None):
def do_unquote(self, url, plus=True, encoding='utf-8', errors='replace'):
def do_urlencode(self, post, doseq=False, safe="", encoding=None, errors=None):
def urljoin(self, domain, url, allow_fragments=True):
def urlparse(self, url, scheme="", allow_fragments=True):
def parse_qs(self, url, keep_blank_values=False, strict_parsing=False):
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, item=Item()):
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 verify_infoLabels_keys(self, item, keys, **AHkwargs):
def get_page_num(self, item, **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):

```

Etiquetas de “finds[‘controls’]” procesadas por AH (suelen prevalecer sobre otras del mismo nombre):

profile	‘DEFAULT’	nombre del perfil por defecto de AH
cnt_tot	20	nº de líneas por página
page_factor	1.0	factor de conversion del last_page real con el que muestra
page_factor_overflow	1.3	factor de overflow para pintar restos de un matches
inicio	5	tiempo máximo loop descargas hasta llenar una pantalla en list_all
custom_pagination	False	activar cuando ninguna opción del paginador funciona
jump_page	False	añade la posibilidad de poder saltar a un nº página
host	self.host	
host_referer	self.host	
post	None	
headers	{}	
forced_proxy_opt	None	opción proxy similar al de canonical
channel_alt	DOMAIN_ALT = 'wolfmax4k'	
url_base64	True	para enlaces ofuscados/encryptados
modo_grafico	config.get_setting('modo_grafico', item.channel, default=True)	
IDIOMAS_TMDB	self.IDIOMAS_TMDB	
btdigg	False	se active en canales que quieres usar los servicios de BTDig
btdigg_quality_control	None	control de calidades para BTDig en canales de separan calidades
force_find_last_page	[]	forzar encontrar last_page [page_min, page_max, 'url' o 'post']
duplicates	[]	Evita duplicados de series/temporadas en list_all
dup_movies	False	si True, también controla duplicados de películas (varias calidades)
dup_list	'title'	title/url
mediatype	"	tipo de contenido
action	"	próxima acción en item.action
tmdb_extended_info	True	si se quiere una 2ª pasa de TMDB para series/episodios
year	False	año por defecto
reverse	False	en categoría, sort inverso (años, ...)
season_TMDB_limit	True	desprecia temporadas que no están en TMDB
add_video_to_videolibrary	True	añadir contenido a la videoteca
videolab_status	True	muestra si el contenido está en la videoteca, con visto/no visto
min_temp	False	si True, solo actualiza la última temporada de videoteca
join_dup_episodes	True	Si webs con varias calidades por episodio, fusiona todas las calidades
host_torrent	self.host	como host en findvideos
host_torrent_referer	self.host	como host en findvideos
sort_findvideos	True	clasificar por tamaño de video
manage_torrents	True	gestionar los servidores “torrent” de forma extendida
premium_skip	"	saltar videos premium con una etiqueta. Adultos

Etiquetas de “elem_json” procesadas por AH:

url	url de la descarga
post:	post de la descarga
headers	headers de la descarga
title	título del contenido. Sustituido por TMDB
thumbnail	foto del contenido. Sustituido por TMDB
quality	calidad del contenido
language	idioma del contenido
mediatype	tipo del contenido: 'movie', 'tvshow', 'episode'
year	año del contenido para TMDB. Sustituido por TMDB
plot	descripción del contenido. Sustituido por TMDB
plot_extend	descripción del contenido, adicional al de TMDB
plot_extend_show	forzar mostrar plot_extend. Unify
broadcast	mensaje que precede a item.contentPlot
extra	indicador de procedencia que se pasa de una función a otra
title_subs	información de temporada, episodio que se añade al título de novedades. Unify
action	siguiente acción a ejecutar, sustituyendo a la estándar: 'list_all', 'season'...
imdb_id	sustituye a código imdb_id
tmdb_id	sustituye a código tmdb_id
tvdb_id	sustituye a código tvdb_id
unify	se activa/desactiva Unify
season	int(nº de temporada)
season_search	nombre normalizado de película/serie para búsquedas
episode	int(nº de episodio)
title_episode	nombre de episodio extendido
context	contexto adicional. Renumbertools, etc
info	información adicional de vídeo
btdig_in_use	acción que usa BTDig
playcount	1/0: visto/no visto
matches	lista de matches procesados para findvideos. Típico de temporadas de canales torrent
server	nombre del servidor
size	tamaño del vídeo. Torrents
torrent_info	información del torrent
torrents_path	path al torrent local
torrent_alt	path al torrent alternativo
password	password de un .rar descargado por torrent
alive	estado del servidor: verificado, no verificado, caído
folder	informa a Kodi si el item tiene folder=True/False
item_org	guarda el item original, formato .tourl
subtitle	path al subtítulo. Puede ser una lista
stime	duración del vídeo. Adultos
premium	vídeo de pago. Adultos
views	nº de visionados. Adultos
channel	nombre de web/canal alternativo en caso de canal multiweb