Vidscraper Documentation

Release 0.5.2

Participatory Culture Foundation

December 27, 2012

CONTENTS

Vidscraper is a clean, simple library for a couple of rather messy issues:

- Retrieving the source video from a “flash-only” website
- Finding out contextual data about a pasted url: title/description/etc

Vidscraper provides a unified api for an issue that requires a lot of one-off scraping.

CHAPTER

ONE

REQUIREMENTS

- json (python2.6) or simplejson
- BeautifulSoup 3.0.8 or 3.2
- feedparser

1.1 Optional

- oauth2 (for some APIs **cough* Vimeo searching *cough** which require authentication)
- nose (for tests)

CHAPTER

TWO

CONTENTS

2.1 Getting Started

2.1.1 Scraping video pages

Most use cases will simply require the auto_scrape function. Usage is incredibly easy:

```
>>> from vidscraper import auto_scrape
>>> video = auto_scrape("http://www.youtube.com/watch?v=J_DV9b0x7v4")
>>> video.title
u'CaramellDansen (Full Version + Lyrics)'
```

That's it! Couldn't be easier. auto_scrape will determine the right *scraping suite* to use for the url you pass in and will use that suite to return a `ScrapedVideo` instance that represents the data associated with the video at that url. If no suites are found which support the url, `CantIdentifyUrl` will be raised.

If you only need certain fields (say you only need the "file_url" and the "title" fields), you can pass those fields in as a second argument:

```
>>> video = auto_scrape(url, fields=['file_url', 'title'])
```

Video fields

If a `ScrapedVideo` is initialized without any fields, then `vidscraper` will assume you want all of the fields for the video. When the `ScrapedVideo` is being loaded, `vidscraper` will maximize the number of requested fields that it fills; occasionally, this may mean that it will make more than one HTTP request. This means that limiting the fields to what you are actually using can save quite a bit of work.

2.1.2 Getting videos for a feed

If you want to get every video for a feed, you can use `vidscraper.auto_feed()`:

```
>>> from vidscraper import auto_feed
>>> results = auto_feed("http://blip.tv/djangocon/rss")
```

This will read the feed at the given url and return a generator which yields `ScrapedVideo` instances for each entry in the feed. The instances will be preloaded with metadata from the feed. In many cases this will fill out all the fields that you need. If you need more, however, you can tell the video to load more data manually:

```
>>> video = results.next()
>>> video.load()
```

(Don't worry - if `vidscraper` can't figure out a way to get more data, it will simply do nothing!)

Note: Because this function returns a generator, the feed will actually be fetched the first time the generator's `next()` method is called.

Crawling an entire feed

`auto_feed()` also supports feed crawling for some suites. You use it like this:

```
>>> from vidscraper import auto_feed
>>> results = auto_feed("http://blip.tv/djangocon/rss", crawl=True)
```

Now, when the generator runs out of results on the first page, it will automatically fetch the next page, and then the next, and so on. This is not for the faint of heart. Depending on the feed you're crawling, you could be there for a while.

2.1.3 Searching video services

It's also easy to run a search on a variety of services that support it. Simply do the following:

```
>>> from vidscraper import auto_search
>>> results = auto_search(['parrot'], exclude_terms=['dead']).values()
```

The search will be run on all suites that support searching, and the results will be returned as a dictionary mapping the suite used to the results for that feed.

2.2 Exceptions

exception `vidscraper.errors.`**`BaseUrlLoadFailure`**
Raised if you can't even load the base url.

exception `vidscraper.errors.`**`CantIdentifyUrl`**
Raised if a url can't be handled by any known *suite*, or if a `Video` is initialized with an incorrect suite.

exception `vidscraper.errors.`**`Error`**
Base error for `vidscraper`.

exception `vidscraper.errors.`**`FieldNotFound`**
Raised if a specific field is not found.

exception `vidscraper.errors.`**`ParsingError`**
Raised if parsing a document with lxml fails.

exception `vidscraper.errors.`**`VideoDeleted`**
Raised if the remote server has deleted the video being scraped.

2.3 Suite API

Vidscraper defines a simple API for "Suites", classes which provide the functionality necessary for scraping video information from a specific video service.

2.3.1 The Suite Registry

`vidscraper.suites.`**`registry`** **= <vidscraper.suites.base.SuiteRegistry object at 0x2360f90>**

An instance of `SuiteRegistry` which is used by `vidscraper` to track registered suites.

class `vidscraper.suites.base.`**`SuiteRegistry`**

A registry of suites. Suites may be registered, unregistered, and iterated over.

`register`(*suite*)

Registers a suite if it is not already registered.

`register_fallback`(*suite*)

Registers a fallback suite, which used only if no other suite succeeds. If no fallback is registered, then CantIdentifyUrl will be raised for unknown videos/feeds.

`suite_for_feed_url`(*url*)

Returns the first registered suite which can handle the `url` as a feed or raises `CantIdentifyUrl` if no such suite is found.

`suite_for_video_url`(*url*)

Returns the first registered suite which can handle the `url` as a video or raises `CantIdentifyUrl` if no such suite is found.

`suites`

Returns a tuple of registered suites.

`unregister`(*suite*)

Unregisters a suite if it is registered.

2.3.2 Built-in Suites

class `vidscraper.suites.`**`BaseSuite`**

This is a base class for suites, demonstrating the API which is expected when interacting with suites. It is not suitable for actual use; some vital methods must be defined on a suite-by-suite basis.

`api_fields` **= set([])**

A set of `Video` fields that this suite can supply optimization.

`apply_video_data`(*video*, *data*)

Stores values from a `data` dictionary on the corresponding attributes of a `Video` instance.

`available_fields`

Returns a set of all of the fields we could possible get from this suite.

`feed_regex` **= None**

A string or precompiled regular expression which will be matched against feed urls to check if they can be handled by this suite.

`get_api_url`(*video*)

Returns the url for fetching API data. May be implemented by subclasses if an API is available.

`get_feed`(*url*, ***kwargs*)

Returns a feed using this suite.

get_feed_description (*feed*, *feed_response*)
: Returns a description of the feed based on the `feed_response`, or `None` if no description can be determined. By default, assumes that the response is a `feedparser` structure and returns a value based on that.

get_feed_entries (*feed*, *feed_response*)
: Returns an iterable of feed entries for a `feed_response` as returned from `get_feed_response()`. By default, this assumes that the response is a `feedparser` structure and tries to return its entries.

get_feed_entry_count (*feed*, *feed_response*)
: Returns an estimate of the total number of entries in this feed, or `None` if that cannot be determined. By default, returns the number of entries in the feed.

get_feed_etag (*feed*, *feed_response*)
: Returns the etag for a `feed_response`, or `None` if no such url can be determined. By default, assumes that the response is a `feedparser` structure and returns a value based on that.

get_feed_guid (*feed*, *feed_response*)
: Returns the guid of the `feed_response`, or `None` if no guid can be determined. By default, assumes that the response is a `feedparser` structure and returns a value based on that.

get_feed_info_response (*feed*, *response*)
: In case the response for the given `feed` needs to do other work on `reponse` to get feed information (title, &c), suites can override this method to do that work. By default, this method just returns the `response` it was given.

get_feed_last_modified (*feed*, *feed_response*)
: Returns the last modification date for the `feed_response` as a python datetime, or `None` if no date can be determined. By default, assumes that the response is a `feedparser` structure and returns a value based on that.

get_feed_response (*feed*, *feed_url*)
: Returns a parsed response for this `feed`. By default, this uses `feedparser` to get a response for the `feed_url` and returns the resulting structure.

get_feed_thumbnail_url (*feed*, *feed_response*)
: Returns the thumbnail URL of the `feed_response`, or `None` if no thumbnail can be found. By default, assumes that the response is a `feedparser` structur4e and returns a value based on that.

get_feed_title (*feed*, *feed_response*)
: Returns a title for the feed based on the `feed_response`, or `None` if no title can be determined. By default, assumes that the response is a `feedparser` structure and returns a value based on that.

get_feed_url (*url*)
: Some suites can handle URLs that are not technically feeds, but can convert them into a feed that is usable. This method can be overidden to do that conversion. By default, this method just returns the original URL.

get_feed_webpage (*feed*, *feed_response*)
: Returns the url for an HTML version of the `feed_response`, or `None` if no such url can be determined. By default, assumes that the response is a `feedparser` structure and returns a value based on that.

get_next_feed_page_url (*feed*, *feed_response*)
: Based on a `feed_response` and a `VideoFeed` instance, generates and returns a url for the next page of the feed, or returns `None` if that is not possible. By default, simply returns `None`. Subclasses must override this method to have a meaningful feed crawl.

get_next_search_page_url (*search*, *search_response*)
: Based on a `VideoSearch` and a `search_response`, generates and returns a url for the next page of the search, or returns `None` if that is not possible. By default, simply returns `None`. Subclasses must override this method to have a meaningful search crawl.

get_oembed_url(*video*)
Returns the url for fetching oembed data. By default, generates an oembed request url based on `oembed_endpoint` or raises `NotImplementedError` if that is not defined.

get_scrape_url(*video*)
Returns the url for fetching scrape data. May be implemented by subclasses if a page scrape should be supported.

get_search(*query*, ***kwargs*)
Returns a search using this suite.

get_search_response(*search*, *search_url*)
Returns a parsed response for the given `search_url`. By default, assumes that the url references a feed and passes the work off to `get_feed_response()`.

get_search_results(*search*, *search_response*)
Returns an iterable of search results for a `VideoSearch` and a `search_response` as returned by `get_search_response()`. By default, assumes that the `search_response` is a `feedparser` structure and passes the work off to `get_feed_entries()`.

get_search_time(*search*, *search_response*)
Returns the amount of time required by the service provider for the suite to execute the search. By default, simply returns `None`.

get_search_total_results(*search*, *search_response*)
Returns an estimate for the total number of search results based on the first response returned by `get_search_response()` for the `VideoSearch`. By default, assumes that the url references a feed and passes the work off to `get_feed_entry_count()`.

get_search_url(*search*)
Returns a url which this suite can use to fetch search results for the given string. Must be implemented by subclasses.

get_video(*url*, ***kwargs*)
Returns a video using this suite.

handles_feed_url(*url*)
Returns `True` if this suite can handle the `url` as a feed and `False` otherwise. By default, this method will check whether the url matches `feed_regex` or raise a `NotImplementedError` if that is not possible.

handles_video_url(*url*)
Returns `True` if this suite can handle the `url` as a video and `False` otherwise. By default, this method will check whether the url matches `video_regex` or raise a `NotImplementedError` if that is not possible.

load_video_data(*video*)
Makes the smallest requests necessary for loading all the missing fields for the `video`. The data is immediately stored on the video instance.

oembed_endpoint = None
A URL which is an endpoint for an oembed API.

oembed_fields
A set of `Video` fields that this suite can supply through an oembed API. By default, this will be empty if `oembed_endpoint` is `None` and a base set of commonly available fields otherwise.

parse_api_error(*exc*)
Parses a **:module:‘urllib‘** exception raised during the API request. If we re-raise an exception, that’s it; otherwise, the dictionary returned will be used to populate the `Video` object.

By default, just re-raises the given exception.

`parse_api_response`(*response_text*)
Parses API response text into a dictionary mapping `Video` field names to values. May be implemented by subclasses if an API is available.

`parse_feed_entry`(*entry*)
Given a feed entry (as returned by `get_feed_entries()`), creates and returns a dictionary containing data from the feed entry, suitable for application via `apply_video_data()`. Must be implemented by subclasses.

`parse_oembed_error`(*exc*)
Parses a **:module:‘urllib‘** exception raised during the OEmbed request. If we re-raise an exception, that’s it; otherwise, the dictionary returned will be used to populate the `Video` object.

By default, just re-raises the given exception.

`parse_oembed_response`(*response_text*)
Parses oembed response text into a dictionary mapping `Video` field names to values. By default, this assumes that the commonly-available fields `title`, `author_name`, `author_url`, `thumbnail_url`, and `html` are available.

`parse_scrape_error`(*exc*)
Parses a **:module:‘urllib‘** exception raised during the scrape request. If we re-raise an exception, that’s it; otherwise, the dictionary returned will be used to populate the `Video` object.

By default, just re-raises the given exception.

`parse_scrape_response`(*response_text*)
Parses scrape response text into a dictionary mapping `Video` field names to values. May be implemented by subclasses if a page scrape should be supported.

`parse_search_result`(*search, result*)
Given a `VideoSearch` instance and a search result (as returned by `get_search_results()`), returns a dictionary containing data from the search result, suitable for application via `apply_video_data()`. By default, assumes that the `result` is a `feedparser` entry and passes the work off to `parse_feed_entry()`.

`scrape_fields` = set([])
A set of `Video` fields that this suite can supply

`video_regex` = None
A string or precompiled regular expression which will be matched against video urls to check if they can be handled by this suite.

2.4 ScrapedVideo API

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX