Vidscraper Documentation

Release 1.0.2

Participatory Culture Foundation

November 14, 2013

Contents

Vidscraper is a library for retrieving information about videos from various sources – video feeds, APIs, page scrapes – combining it, and presenting it in a unified manner, all as efficiently as possible.

Vidscraper comes with built-in support for popular video sites like blip, vimeo, ustream, and youtube, as well support for generic RSS feeds with feedparser.

CHAPTER 1

Quick example

```
>>> import vidscraper
>>> video = vidscraper.auto_scrape('http://www.youtube.com/watch?v=PMpu8jH1LE8')
>>> video.title
u"The Magic Roundabout - Ermintrude's Folly"
>>> video.description
u"Ermintrude's been at the poppies again, but it's Dougal who ends up high as a kite!"
>>> video.user
u'nickhirst999'
>>> video.guid
'http://gdata.youtube.com/feeds/api/videos/PMpu8jH1LE8'
```

1.1 Command line

vidscraper also comes with a command line utility allowing you to get video metadata from the command line. The example above could look like this:

```
$ vidscraper video http://www.youtube.com/watch?v=PMpu8jH1LE8 \
  --fields=title,description,user,guid
Scraping http://www.youtube.com/watch?v=PMpu8jH1LE8...
{
  "description": "Ermintrude's been at the poppies again, but it's Dougal who ends up high as a kite
  "fields": [
    "title",
    "description",
    "user",
    "guid"
  ],
  "guid": "http://gdata.youtube.com/feeds/api/videos/PMpu8jH1LE8",
  "title": "The Magic Roundabout - Ermintrude's Folly",
  "url": "http://www.youtube.com/watch?v=PMpu8jH1LE8",
  "user": "nickhirst999"
}
```

CHAPTER 2

Project links

code https://github.com/pculture/vidscraper/

docs http://vidscraper.readthedocs.org/

bugtracker http://bugzilla.pculture.org/

code https://github.com/pculture/vidscraper/

irc #vidscraper on irc.freenode.net

build status

2.1 Requirements

- Python 2.6+
- BeautifulSoup 4.0.2+
- feedparser 5.1.2+
- python-requests 0.13.0+ (But less than 1.0.0!)

2.1.1 Optional

- requests-oauth 0.4.1+ (for some APIs **cough* Vimeo searching *cough** which require authentication)
- lxml 2.3.4+ (recommended for BeautifulSoup; assumed parser for test results.)
- unittest2 0.5.1+ (for tests)
- mock 0.8.0+ (for tests)
- tox 1.4.2+ (for tests)

2.2 User Guide

2.2.1 Getting Started

Scraping video pages

Most use cases will simply require the `auto_scrape()` function.

```
>>> 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 pull down metadata from all the places it can figure out based on the url you entered and return a `Video` instance loaded with that data.

If `vidscraper` doesn't know how to fetch data for that url – for example, if you try to scrape google.com (which isn't a video page) – `UnhandledVideo` will be raised.

Limiting metadata

Videos can have metadata pulled from a number of sources - for example, a page scrape, an OEmbed API, and a service-specific API. When loading video data, `vidscraper` will query as many of these services as it needs to provide the data you ask for.

So, if you only need certain pieces of metadata (say the title and description of a video), you can pass those fields to `auto_scrape()` and potentially save HTTP requests:

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

See Also:

`Video`

Getting videos for a feed

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

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

This will read the feed at the given url and return a generator which yields `Video` 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 = feed.next()
>>> video.load()
```

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

The feed instance is a lazy generator - it won't make any HTTP requests until you call `next()` the first time. It will only make a second request once you've gotten to the bottom of the first page.

Not crawling a whole feed

By default, `auto_feed()` will try to crawl through the entire feed. Depending on the feed you're crawling, you could be there for a while. If you're pressed for time (or bandwidth) you can limit the number of videos you pull down:

```
>>> from vidscraper import auto_feed
>>> feed = auto_feed("http://blip.tv/djangocon/rss")
>>> len(list(feed))
117
>>> feed = auto_feed("http://blip.tv/djangocon/rss", max_results=20)
>>> len(list(feed))
20
```

Searching video services

It's also easy to run a search on a variety of services with `auto_search()`:

```
>>> from vidscraper import auto_search
>>> searches = auto_search('parrot -dead', max_results=20)
>>> searches
[<vidscraper.suites.blip.Search object at 0x10b490f90>,
 <vidscraper.suites.youtube.Search object at 0x10b49f090>]
```

You'll get back a list of search iterables for suites which support the search parameters. These have the same behavior in terms of loading new pages that you see in the feed iterator.

```
>>> video = searches[0].next()
>>> video.title
u"Episode 57: iMovie HD '06, Part II"
```

2.3 API Documentation

2.3.1 Main Interface

The 5 functions given here should handle most use cases.

vidscraper.**auto_scrape**(*url*, *fields=None*, *api_keys=None*)

Returns a `Video` instance with data loaded.

Parameters

- **url** – A video URL. Video website URLs generally work; more obscure urls (like API urls) might work as well.
- **fields** – A list of fields to be fetched for the video. Limiting this may decrease the number of HTTP requests required for loading the video.

  See Also:

  Limiting metadata
- **api_keys** – A dictionary of API keys for various services. Check the documentation for each `suite` to find what API keys they may want or require.

Raises `UnhandledVideo` if no `suite` can be found which handles the video.

vidscraper.**auto_feed**(*self*, *url*, *last_modified=None*, *etag=None*, *start_index=1*, *max_results=None*, *video_fields=None*, *api_keys=None*)
For each registered suite, calls get_feed() with the given parameters, until a suite returns a feed instance.

Returns An instance of a specific suite's feed_class with no data loaded.

Raises UnhandledFeed if no registered suites know how to handle this url.

vidscraper.**auto_search**(*self*, *query*, *order_by='relevant'*, *start_index=1*, *max_results=None*, *video_fields=None*, *api_keys=None*)
For each registered suite, calls get_search() with the given parameters.

Returns a list of iterators over search results for suites which support the given parameters.

vidscraper.**handles_video**()
Returns True if any registered suite can make a video with the given parameters, and False otherwise.

Note: This does all the work of creating a video, then discards it. If you are going to use a video instance if one is created, it would be more efficient to use auto_scrape() directly.

---

vidscraper.**handles_feed**()
Returns True if any registered suite can make a feed with the given parameters, and False otherwise.

Note: This does all the work of creating a feed, then discards it. If you are going to use a feed instance if one is created, it would be more efficient to use auto_feed() directly.

2.3.2 Exceptions

exception vidscraper.exceptions.**VidscraperError**
Base error for vidscraper.

exception vidscraper.exceptions.**UnhandledVideo**
Raised by VideoLoaders and *suites* if a given video can't be handled.

exception vidscraper.exceptions.**UnhandledFeed**
Raised if a feed can't be handled by a *suite* or BaseFeed subclass.

exception vidscraper.exceptions.**UnhandledSearch**
Raised if a search can't be handled by a *suite* or BaseSearch subclass.

exception vidscraper.exceptions.**InvalidVideo**
Raised if a video is found to be invalid in some way after data has been collected on it.

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

2.3.3 Suite API

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

The Suite Registry

vidscraper.suites.**registry** = **<vidscraper.suites.base.SuiteRegistry object at 0x3a5dad0>**
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.

get_feed(*url*, *last_modified=None*, *etag=None*, *start_index=1*, *max_results=None*, *video_fields=None*, *api_keys=None*)

For each registered suite, calls get_feed() with the given parameters, until a suite returns a feed instance.

Returns An instance of a specific suite's feed_class with no data loaded.

Raises UnhandledFeed if no registered suites know how to handle this url.

get_searches(*query*, *order_by='relevant'*, *start_index=1*, *max_results=None*, *video_fields=None*, *api_keys=None*)

For each registered suite, calls get_search() with the given parameters.

Returns a list of iterators over search results for suites which support the given parameters.

get_video(*url*, *fields=None*, *api_keys=None*, *require_loaders=True*)

For each registered suite, calls get_video() with the given url, fields, and api_keys, until a suite returns a Video instance.

Parameters **require_loaders** – Changes the behavior if no suite is found which handles the given parameters. If True (default), UnhandledVideo will be raised; otherwise, a video will returned with the given url and fields, but it will not be able to load any additional data.

Raises UnhandledVideo if require_loaders is True and no registered suite returns a video for the given parameters.

Returns Video instance with no data loaded.

handles_feed(**args*, ***kwargs*)

Returns True if any registered suite can make a feed with the given parameters, and False otherwise.

Note: This does all the work of creating a feed, then discards it. If you are going to use a feed instance if one is created, it would be more efficient to use get_feed() directly.

---

handles_video(**args*, ***kwargs*)

Returns True if any registered suite can make a video with the given parameters, and False otherwise.

Note: This does all the work of creating a video, then discards it. If you are going to use a video instance if one is created, it would be more efficient to use get_video() directly.

---

register(*suite*)

Registers a suite if it is not already registered.

register_fallback(*suite*)

Registers a fallback suite, which is tried only if no other suite successfully handles a given video, feed, or search.

suites

Returns a tuple of registered suites. If a fallback is registered, it will always be at the end of this tuple.

unregister(*suite*)

Unregisters a suite if it is registered.

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.

`available_fields`

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

`feed_class = None`

A `BaseFeed` subclass that will be used to parse feeds for 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_feed`(*url*, **args*, ***kwargs*)

Returns an instance of `feed_class`, which should be a subclass of `BaseFeed`.

Raises `UnhandledFeed` if `feed_class` is None.

`get_search`(**args*, ***kwargs*)

Returns an instance of `search_class`, which should be a subclass of `BaseSearch`.

Raises `UnhandledSearch` if `search_class` is None.

`get_video`(*url*, *fields=None*, *api_keys=None*)

Returns a video using this suite's loaders. This instance will not have data loaded.

Parameters

- **url** – A video URL. Video website URLs generally work; more obscure urls (like API urls) might work as well.
- **fields** – A list of fields to be fetched for the video. Limiting this may decrease the number of HTTP requests required for loading the video.

  See Also:

  Limiting metadata
- **api_keys** – A dictionary of API keys for various services. Check the documentation for each `suite` to find what API keys they may want or require.

Raises `UnhandledVideo` if none of this suite's loaders can handle the given url and api keys.

`handles_feed`(**args*, ***kwargs*)

Returns `True` if this suite can make a feed with the given parameters, and `False` otherwise.

Note: This does all the work of creating a feed, then discards it. If you are going to use a feed instance if one is created, it would be more efficient to use `get_feed()` directly.

`handles_search`(**args*, ***kwargs*)

Returns `True` if this suite can make a search with the given parameters, and `False` otherwise.

Note: This does all the work of creating a search, then discards it. If you are going to use a search instance if one is created, it would be more efficient to use `get_search()` directly.

`handles_video`(**args*, ***kwargs*)

Returns `True` if this suite can make a video with the given parameters, and `False` otherwise.

Note: This does all the work of creating a video, then discards it. If you are going to use a video instance if one is created, it would be more efficient to use `get_video()` directly.

---

loader_classes = ()
A list or tuple of `VideoLoader` classes which will be used to populate videos with data. These loaders will be run in the order they are given, so it's a good idea to order them by the effort they would require; for example, OEmbed should generally come first, since the response is small and easy to parse compared to, say, a page scrape.

See Also:

`Video.run_loaders()`

search_class = None
A `BaseSearch` subclass that will be used to run searches for this suite.

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.

2.3.4 Video API

class vidscraper.videos.**Video**(*url*, *loaders=None*, *fields=None*)
This is the class which should be used to represent videos which are returned by suite scraping, searching and feed parsing.

Parameters

- **url** – The "pasted" url for the video.
- **loaders** – An iterable of `VideoLoader` instances that will be used to load video data.
- **fields** – A list of fields which should be fetched for the video. This will be used to optimize the fetching process. Other fields will not populated, even if the data is available.

link = None
The canonical link to the video. This may not be the same as the url used to initialize the video.

guid = None
A (supposedly) global identifier for the video

index = None
Where the video was in the feed/search

title = None
The video's title.

description = None
A text or html description of the video.

publish_datetime = None
A python datetime indicating when the video was published.

files = None
A list of `VideoFile` instances representing all the possible files for this video.

flash_enclosure_url = None
"Crappy enclosure link that doesn't actually point to a url.. the kind crappy flash video sites give out when they don't actually want their enclosures to point to video files."

embed_code = None
The actual embed code which can be used for displaying the video in a browser.

thumbnail_url = None
The url for a thumbnail of the video.

user = None
The username associated with the video.

user_url = None
The url associated with the video's user.

tags = None
A list of tag names associated with the video.

license = None
A URL to a description of the license the Video is under (often Creative Commons)

is_embeddable = None
Whether the video is embeddable? (Youtube, Vimeo)

missing_fields
Returns a list of fields which have been requested but which have not been filled with data.

load()
If the video hasn't been loaded before, runs the loaders and populates the video's `fields`.

get_best_loaders()
Returns a list of loaders from `loaders` which can be used in combination to fill all missing fields - or as many of them as possible.

This will prefer the first listed loaders and will prefer small combinations of loaders, so that the smallest number of smallest possible responses will be fetched.

run_loaders()
Runs `get_best_loaders()` and then gets data from each loader.

items()
Iterator over (field, value) for requested fields.

serialize()
Serializes the video as a python dictionary containing the original url and fields used to initialize the video, as well as the value of each field on the video. Since loaders are intended to be provided by suites and include sensitive information (api keys), they are not serialized.

classmethod **deserialize**(*data*, *api_keys=None*)
Given a data dictionary such as would be provided by `serialize()` and, optionally, api keys, constructs a `Video` instance for the url and fields in the data, with field values prepopulated from the dictionary.

Parameters

- **data** – A dictionary as would be provided by `serialize()`.
- **api_keys** – `None`, or a dictionary of API keys to instantiate the deserialized video with.

get_file(*preferred_mimetypes=('video/webm', 'video/ogg', 'video/mp4')*)
Returns the preferred file from the files for this video. Vidscraper prefers open formats and well-compressed formats. If no file mimetypes are known, the first file will be returned.

class vidscraper.videos.**VideoFile**(*url*, *expires=None*, *length=None*, *width=None*, *height=None*, *mime_type=None*)
Represents a video file hosted somewhere. The only required attribute is the file's `url`. There are also several optional metadata attributes, which represent what is claimed about the video by the data provider, not necessarily what is actually true about the video.

url = None
The URL of this video file.

expires = None
When the URL for this file expires, if at all.

length = None
The size of the file, in bytes.

width = None
The width of the video, in pixels.

height = None
The height of the video, in pixels.

mime_type = None
The MIME type of the video.

serialize()
Serializes the VideoFile as a python dictionary.

classmethod deserialize(*data*)
Given a data dictionary such as would be provided by serialize(), constructs a VideoFile instance.

class vidscraper.videos.**VideoLoader**(*url*, *api_keys=None*)
This is a base class for objects that fetch data for a video, for example from an API or a page scrape.

Parameters

- **url** – The "pasted" url for which data should be loaded.
- **api_keys** – A dictionary of API keys which may be needed to load data with this loader.

fields = set([])
A set of fields this loader believes it can provide.

url_format = None
A format string which, paired with url_data, returns a suitable url for this loader to fetch data from.

timeout = 3
The number of seconds before this loader times out. See python-requests documentation for more information.

headers = {'User-Agent': 'Mozilla/5.0 (X11; Linux) Safari/536.10 vidscraper/(1, 0, 2)'}
Extra headers to set on the requests for this loader. See python-requests documentation for more information.

get_url_data(*url*)
Parses the url into data which can be used to construct a url this loader can use to get data.

Raises UnhandledVideo if the url isn't handled by this loader.

get_url()
Returns a url which can be fetched to get a response that this loader can process into data.

get_headers()
Returns a dictionary of headers which will be added to the request. By default, this is a copy of headers.

get_request_kwargs()
Returns the kwargs used for making an HTTP request for this loader (with python-requests).

get_video_data(*response*)
Parses the given response and returns a data dictionary for populating a Video instance. By default, returns an empty dictionary.

class vidscraper.videos.**OEmbedLoaderMixin**

Mixin to provide basic OEmbed functionality. Subclasses need to provide an endpoint, define a get_url_data method, and provide a url_format - for the video URL, not the oembed API URL.

This is provided as a mixin rather than a subclass of `VideoLoader` so that it can be used on top of any class or mixin that overrides `VideoLoader.get_url()`.

endpoint = None

The endpoint for the OEmbed API.

class vidscraper.videos.**VideoIterator**(*start_index=1*, *max_results=None*, *video_fields=None*, *api_keys=None*)

Generic base class for iterating over groups of videos spread across multiple urls - for example, an rss feed, an api response, or a video list page.

Parameters

- **start_index** (*integer >= 1*) – The index of the first video to return. Default: 1.
- **max_results** – The maximum number of videos to return. If this is `None` (the default), as many videos as possible will be returned.
- **video_fields** – A list of fields to be fetched for each video in the iterator. Limiting this may decrease the number of HTTP requests required for loading video data.

  See Also:

  Limiting metadata
- **api_keys** – A dictionary of API keys for various services. Check the documentation for each `suite` to find what API keys they may want or require.

per_page = None

Describes the number of videos expected on each page. This should be set whether or not the number of videos per page can be controlled.

page_url_format = None

A format string which will be used to build page urls for this iterator. This should use the {} format described under str.format() in the python docs: http://docs.python.org/library/stdtypes.html#str.format

timeout = 3

The number of seconds before this loader times out. See python-requests documentation for more information.

headers = {'User-Agent': 'Mozilla/5.0 (X11; Linux) Safari/536.10 vidscraper/(1, 0, 2)'}

Extra headers to set on the requests for this loader. See python-requests documentation for more information.

load()

Loads a response if one is not already loaded and tries to extract data from it with `data_from_response()`.

get_response_items(*response*)

Returns an iterable of unparsed items for the response.

get_video_data(*item*)

Parses a single item for the feed and returns a data dictionary for populating a `Video` instance. By default, returns an empty dictionary. Raises `InvalidVideo` if the item is found to be invalid in some way; this causes the item to be ignored.

get_page_url(*page_start*, *page_max*)

Builds and returns a url for a page of the source by putting the results of `get_page_url_data()` into `page_url_format`.

get_page_url_data(*page_start*, *page_max*)
Returns a dictionary which will be combined with `page_url_format` to build a page url.

get_headers()
Returns a dictionary of headers which will be added to the request. By default, this is a copy of `headers`.

get_request_kwargs()
Returns the kwargs used for making an HTTP request for this feed.

get_page(*page_start*, *page_max*)
Given a start and maximum size for a page, fetches and returns a response for that page. The response could be a feedparser dict, a parsed json response, or even just an html page.

data_from_response(*response*)
Given a response as returned from `get_page()`, returns a dictionary of metadata about this iterator. By default, returns an empty dictionary.

class `vidscraper.videos.`**FeedparserVideoIteratorMixin**
Overrides the `get_page()`, `data_from_response()` and `get_response_items()` to use `feedparser`. `get_video_data()` must still be implemented by subclasses.

class `vidscraper.videos.`**BaseFeed**(*url*, *last_modified=None*, *etag=None*, ***kwargs*)
Represents a list of videos which can be found at a certain url. The source could easily be an RSS feed, an API response, or a video list page.

In addition to the parameters for `VideoIterator`, this class takes the following arguments:

Parameters

- **url** – A url representing a feed page.
- **last_modified** – The last known modification date for the feed. This can be sent to the service provider to try to short-circuit fetching and/or loading a feed whose contents are already known.
- **etag** – An etag which can be sent to the service provider to try to short-circuit fetching a feed whose contents are already known.

  See Also:

  http://en.wikipedia.org/wiki/HTTP_ETag

Raises `UnhandledFeed` if the url can't be handled by the class being instantiated.

`BaseFeed` also supports the following "fields", which are populated with `data_from_response()`. Fields which have not been populated will be `None`.

video_count
The estimated number of videos for the feed.

last_modified
A python datetime representing when the feed was last changed. Before loading the feed, this will be equal to the `last_modified` date the `BaseFeed` was instantiated with.

etag
A marker representing a feed's current state. Before loading the feed, this will be equal to the `etag` the `BaseFeed` was instantiated with.

description
A description of the feed.

webpage
The url for an html, human-readable version of the feed.

title
The title of the feed.

thumbnail_url
A URL for a thumbnail representing the whole feed.

guid
A unique identifier for the feed.

get_url_data(*url*)
Parses the url into data which can be used to construct page urls.

Raises UnhandledFeed if the url isn't handled by this feed.

class vidscraper.videos.**BaseSearch**(*query*, *order_by='relevant'*, ***kwargs*)
Represents a search on a video site. In addition to the parameters for VideoIterator, this class takes the following arguments:

Parameters

- **query** – The raw string for the search.
- **order_by** – The ordering to apply to the search results. If a suite does not support the given ordering, it will return an empty list. Possible values: relevant, latest, popular. Values may be prefixed with a "-" to indicate descending ordering. Default: relevant.

Raises UnhandledSearch if the class doesn't support the given parameters.

BaseSearch also supports the following "fields", which are populated with data_from_response(). Fields which have not been populated will be None.

video_count
The estimated number of total videos for this search.

order_by_map = {'relevant': 'relevant'}
Dictionary mapping our order_by options (relevant, latest, and popular) to the service's equivalent term. If an order_by option is not in this dictionary, it is assumed not to be supported by the service.

2.4 Release notes

2.4.1 1.0.2 release notes

- Corrected package data installation.
- Added basic sanity-checking for video urls.

2.4.2 1.0.1 release notes

- Fixed parsing bugs in Youtube, Blip, and generic suites.
- Upped scraped thumbnail size for Vimeo.
- Added Youtube developer key support.

2.4.3 1.0.0 release notes

1.0.0 introduces the following changes, many of which are backwards-incompatible with 0.5.

- `VideoFeed` and `VideoSearch` have been renamed to `BaseFeed` and `BaseSearch`, respectively.
- `Video`, `BaseFeed`, and `BaseSearch` are now found in `vidscraper.videos`. They can no longer be imported from `vidscraper` or `vidscraper.suites`.
- `BaseFeed` and `BaseSearch` now fully implement the python generator API.
- `BaseFeed` and `BaseSearch` now support fetching specific slices of videos instead of fetching a certain number of pages.
- `VideoLoaders` were introduced as a more generic way to represent getting data for a single video.
- `vidscraper.errors` was renamed to `vidscraper.exceptions`, and unused exceptions were removed. `CantIdentifyUrl` was split into two exceptions (`UnhandledVideo` and `UnhandledFeed`) and `Error` is now `VidscraperError`.
- `vidscraper.handles_video_url()` and `vidscraper.handles_feed_url()` were renamed to `vidscraper.handles_video()` and `vidscraper.handles_feed()`, respectively. They now accept the same parameters as `vidscraper.auto_scrape()` and `vidscraper.auto_feed()`.
- Multiple `VideoFile`s are now made available for `Video` instances, rather than having a single set of fields on the `Video` class.
- Removed `vidscraper.utils.http` since the single function there was unrelated to HTTP, was only used by a single suite, and was of questionable usefulness period.
- Added support for testing with tox.
- Added vidscraper-cmd for shell access to the api.
- Replaced video pickling support with simple video serialization.
- Started using python-requests where possible.
- `auto_search()` now returns a simple list of searches rather than a dictionary mapping suites to searches.

CHAPTER 3

Indices and tables

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

Python Module Index

V