toFeed Documentation

Release 0.1

cryzed

May 30, 2014

toFeed

toFeed aims to provide syndication feeds for websites that don't.

What toFeed does at its core is scraping websites, converting the gathered data into syndication feed formats, such as RSS or Atom, and exposing the generated feeds to news aggregators through a web service. It grew out of my desire to be able to immediately see news from sites I regularly visit and filter them according to my own preferences.

toFeed relies on third-party modules such as BeautifulSoup, Jinja2 and Flask to scrape the websites and generate as well as expose the feeds. Of course that doesn't mean that you are limited to these modules, writing your own *adapter* is easy and you are free to use whatever modules you want to do so. The decision to use BeautifulSoup instead of lxml.html was primarily made to avoid binary dependencies which would make the package less portable and harder to install for end users. Another reason was that I'm simply more familiar and comfortable working with BeautifulSoup.

CHAPTER 1

Usage

You can either run toFeed locally on your own PC or externally on a server. It is recommended to use virtualenv in either case.

If you are planning on running toFeed locally, simply execute the main module and the toFeed service should start running on your localhost and expose the routes to your adapters from there.

Alternatively, if you are interested in setting up an external toFeed instance, I recommend using Heroku, which allows you to do so at no cost at all. Simply follow their Getting Started with Python on Heroku guide from the Declare process types with Procfile section onwards.

CHAPTER 2

Modules

2.1 Adapters

Adapters are the heart of toFeed. They take care of retrieving, scraping and generating the syndication feeds for sites you wish to subscribe to.

Adapters can be grouped together in modules. Such a module must contain a `ROUTE` top-level constant, which is used as the first part of the URL you give your news aggregator to retrieve the content. The complete URL is built by appending the `ROUTE` class variable of the specific adapter.

For example the built-in adapters include an adapter specific to parsing the Twitter timeline widget data. In the `twitter` module the `ROUTE` equals `twitter` and the adapter's `ROUTE` class variable equals `timelineWidget`. That means this adapter would be reachable at: `twitter/timelineWidget`.

Notable also is that it's the primary adapter of this module, meaning instead of having to write `twitter/timelineWidget`, simply using `twitter` i.e. the `ROUTE` of the module itself is sufficient.

When opening a route you can pass in arguments and keyword arguments which the adapters can use to implement certain functionality, for example limiting the title length of feed items. This is simply done via GET parameters. A `?key=value` pair would be interpreted as a keyword argument. A key without a value, i.e. `?test& ...` would be a simple positional argument. If an adapter needs to accept such parameters you need to define them in the constructor.

It is good practice to accept `**kwargs` in your adapter's constructor and forward them to the base adapter's constructor, this way certain functionality that makes sense for every adapter can be implemented and utilized. Currently this is limited to manually setting the cache timeout, e.g. `twitter?cache_timeout=300`.

class `tofeed.adapters.`**`Adapter`**(*cache_timeout='120'*)

The base adapter class. Each adapter needs to inherit from this class.

Variables

- **ROUTE** (*str*) – The route leading to the adapter; must be set by the inheriting adapter.
- **PRIMARY** (*bool*) – If set to `True`, the adapter will be recognized as the module's primary adapter and be directly accessible via the module's route in addition to its own route. If this is the case, implementing the `ROUTE` class variable is optional.
- **CACHE_TIMEOUT** (*str*) – The default time to cache the content returned by the adapter's implementation of `to_feed()`.

Note Parameters received by the constructor are strings and must be handled accordingly.

Parameters **cache_timeout** (*str*) – The time to cache the content returned by the adapter's implementation of `to_feed()`.

to_feed()
Must be implemented by the inheriting adapter.

Return type str

Returns The generated feed content.

2.2 Utilities

The utilities package and its modules provide functions and helpers for transforming the scraped data.

`tofeed.utilities.`**`shorten_to_title`**(*string*, *length*, *separator=' '*, *appendix='...'*)
Use the string to create a human readable title.

The function searches for the last occurrence of the separator string within the range of the string limited by the length parameter and appends the appendix.

Parameters

- **string** (*str*) – The string to create the title out of
- **length** (*int*) – The approximate length of the title. The length is only approximate to this value, because it's possible that the next separator string encountered may lie further ahead.
- **separator** (*str*) – The character that separates words in the text. Usually of course a space, this should only have to be changed in very rare cases.
- **appendix** (*str*) – The string to append to the end of the title

Returns The title created from the string

2.2.1 Spoon

Helper functions for modifying `BeautifulSoup` objects.

`tofeed.utilities.spoon.`**`absolutize_references`**(*base_url*, *tag*, *attributes=['href', 'src']*, *recursive=True*)
Turns references found within the tag's attributes absolute.

Parameters

- **base_url** (*str*) – The base URL used to absolutize the references
- **tag** (*bs4.element.Tag*) – The tag to absolutize references in
- **attributes** (*list*) – The attributes containing the URLs that should be made absolute
- **recursive** (*bool*) – If true the tag and all its sub tags will be searched, else only the tag and its direct descendants will be searched.

`tofeed.utilities.spoon.`**`collapse_tag`**(*tag*)
Replaces the tag's descendants with their strings.

Parameters **tag** (*bs4.element.Tag*) – The tag to collapse.

`tofeed.utilities.spoon.`**`convert_newlines`**(*tag*, *recursive=True*)
Replaces newline characters found in the tag's strings with line break tags.

Parameters

- **tag** (*bs4.element.Tag*) – The tag to convert newline characters in.

- **recursive** (*bool*) – If true the tag and all its sub tags will be searched, else only the tag and its direct descendants will be searched.

`tofeed.utilities.spoon.`**`new_string`** **= <bound method BeautifulSoup.new_string of >**

Shortcut for `BeautifulSoup.new_string()`. This allows using the bound methods of the BeautifulSoup class without having to instantiate it manually.

`tofeed.utilities.spoon.`**`new_tag`** **= <bound method BeautifulSoup.new_tag of >**

Shortcut for `BeautifulSoup.new_tag()`. This allows using the bound methods of the BeautifulSoup class without having to instantiate it manually.

`tofeed.utilities.spoon.`**`replace_string_with_tag`**`(`*tag*, *string*, *replacement*, *recursive=True*`)`

Replaces all occurrences of string within the tag's strings with the replacement tag.

Parameters

- **tag** (*bs4.element.Tag*) – The tag to replace strings in
- **string** (*str*) – The string to replace
- **bs4.element.Tag replacement** (*str*) – The tag replacing the string
- **recursive** (*bool*) – If true the tag and all its sub tags will be searched, else only the tag and its direct descendants will be searched.

CHAPTER 3

Indices and tables

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

Python Module Index