micawber Documentation

Release 0.3.4

charles leifer

Feb 14, 2018

Contents

A small library for extracting rich content from urls.

https://github.com/coleifer/micawber

micawber supplies a few methods for retrieving rich metadata about a variety of links, such as links to youtube videos. micawber also provides functions for parsing blocks of text and html and replacing links to videos with rich embedded content.

CHAPTER 1

examples

here is a quick example:

```
import micawber

# load up rules for some default providers, such as youtube and flickr
providers = micawber.bootstrap_basic()

providers.request('http://www.youtube.com/watch?v=54XHDUOHuzU')

# returns the following dictionary:
{
    'author_name': 'pascalbrax',
    'author_url': u'http://www.youtube.com/user/pascalbrax'
    'height': 344,
    'html': u'<iframe width="459" height="344" src="http://www.youtube.com/embed/
↪54XHDUOHuzU?fs=1&feature=oembed" frameborder="0" allowfullscreen></iframe>',
    'provider_name': 'YouTube',
    'provider_url': 'http://www.youtube.com/',
    'title': 'Future Crew - Second Reality demo - HD',
    'type': u'video',
    'thumbnail_height': 360,
    'thumbnail_url': u'http://i2.ytimg.com/vi/54XHDUOHuzU/hqdefault.jpg',
    'thumbnail_width': 480,
    'url': 'http://www.youtube.com/watch?v=54XHDUOHuzU',
    'width': 459,
    'version': '1.0',
}

micawber.parse_text('this is a test:\nhttp://www.youtube.com/watch?v=54XHDUOHuzU',␣
↪providers)

# returns the following string:
this is a test:
<iframe width="459" height="344" src="http://www.youtube.com/embed/54XHDUOHuzU?fs=1&
↪feature=oembed" frameborder="0" allowfullscreen></iframe>
```

```
micawber.parse_html('<p>http://www.youtube.com/watch?v=54XHDUOHuzU</p>', providers)

# returns the following html:
<p><iframe width="459" height="344" src="http://www.youtube.com/embed/54XHDUOHuzU?
↪fs=1&amp;feature=oembed" frameborder="0" allowfullscreen="allowfullscreen"></iframe>
↪</p>
```

check out the *getting started* for more examples

CHAPTER 2

integration with web frameworks

- *flask*
- *django*

Contents:

2.1 Installation

First, you need to install micawber

There are a couple of ways:

2.1.1 Installing with pip

```
pip install micawber

or

pip install -e git+https://github.com/coleifer/micawber.git#egg=micawber
```

2.1.2 Installing via git

```
git clone https://github.com/coleifer/micawber.git
cd micawber
python setup.py test
sudo python setup.py install
```

Adding to your Django Project

After installing, adding django-utils to your projects is a snap. Simply add it to your projects' INSTALLED_APPs and run 'syncdb':

```
# settings.py
INSTALLED_APPS = [
    ...
    'micawber.contrib.mcdjango'
]
```

2.2 Getting Started

If you want the dead simple get-me-up-and-running, try the following:

```
>>> import micawber
>>> providers = micawber.bootstrap_embedly() # may take a second
>>> print micawber.parse_text('this is a test:\nhttp://www.youtube.com/watch?
↪v=54XHDUOHuzU', providers)
this is a test:
<iframe width="640" height="360" src="http://www.youtube.com/embed/54XHDUOHuzU?fs=1&
↪feature=oembed" frameborder="0" allowfullscreen></iframe>
```

Using django? Add `micawber.contrib.mcdjango` to your `INSTALLED_APP`, then in your templates:

```
{% load micawber_tags %}
{# show a flash player for the youtube video #}
{{ "http://www.youtube.com/watch?v=mQEWI1cn7HY"|oembed }}
```

Using flask? Use the `add_oembed_filters` function to register two jinja template filters, `oembed` and `extract_oembed`:

```
from flask import Flask
from micawber.providers import bootstrap_basic
from micawber.contrib.mcflask import add_oembed_filters

app = Flask(__name__)

oembed_providers = bootstrap_basic()
add_oembed_filters(app, oembed_providers)
```

```
{# show a flash player for the youtube video #}
{{ "http://www.youtube.com/watch?v=mQEWI1cn7HY"|oembed() }}
```

2.2.1 Overview

micawber is rather simple. It is built to use the oembed spec, which is designed for converting URLs into rich, embeddable content. Many popular sites support this, including youtube and flickr. There is also a 3rd-party service called embedly that can convert many types of links into rich content.

micawber was designed to make it easy to integrate with these APIs. There are three main concepts to grok when using micawber:

- *Provider* objects

- *ProviderRegistry* objects
- *parsers* module and its functions

2.2.2 Providers

Providers are used to convert URLs into rich metadata. They have an endpoint associated with them and can have any number of arbitrary URL parameters (such as API keys) which are used when making API requests.

Example:

```
from micawber.providers import Provider

youtube = Provider('http://www.youtube.com/oembed')
youtube.request('http://www.youtube.com/watch?v=nda_OSWeyn8')
```

The above code returns a dictionary containing metadata about the requested video, including the markup for an embeddable player:

```
{'author_name': u'botmib',
 'author_url': u'http://www.youtube.com/user/botmib',
 'height': 344,
 'html': u'<iframe width="459" height="344" src="http://www.youtube.com/embed/nda_
↪OSWeyn8?fs=1&feature=oembed" frameborder="0" allowfullscreen></iframe>',
 'provider_name': u'YouTube',
 'provider_url': u'http://www.youtube.com/',
 'thumbnail_height': 360,
 'thumbnail_url': u'http://i3.ytimg.com/vi/nda_OSWeyn8/hqdefault.jpg',
 'thumbnail_width': 480,
 'title': u'Leprechaun in Mobile, Alabama',
 'type': u'video',
 'url': 'http://www.youtube.com/watch?v=nda_OSWeyn8',
 'version': u'1.0',
 'width': 459}
```

More information can be found in the *Provider* API docs.

2.2.3 ProviderRegistry

The *ProviderRegistry* is a way of organizing lists of providers. URLs can be requested from the registry and if *any* provider matches it will be used, otherwise a `ProviderException` will be raised.

The `ProviderRegistry` also supports an optional simple caching mechanism.

Here is an excerpt from the code from the *micawber.providers.bootstrap_basic()* function, which is handy for grabbing a `ProviderRegistry` with a handful of basic providers pre-populated:

```
def bootstrap_basic(cache=None, registry=None, **params):
    pr = registry or ProviderRegistry(cache)
    pr.register('http://\S*?flickr.com/\S*', Provider('http://www.flickr.com/services/
↪oembed/'))
    pr.register('http://\S*.youtu(\.be|be\.com)/watch\S*', Provider('http://www.
↪youtube.com/oembed'))
    pr.register('http://www.hulu.com/watch/\S*', Provider('http://www.hulu.com/api/
↪oembed.json'))
    return pr
```

As you can see, the *register()* method takes two parameters, a regular expression for valid URLs and a `Provider` instance.

You can use helper functions to get a populated registry:

- *bootstrap_basic()*
- *bootstrap_embedly()*
- *bootstrap_noembed()*
- *bootstrap_oembedio()*

The `bootstrap_embedly`, `bootstrap_noembed` and `bootstrap_oembedio` functions make a HTTP request to the API server asking for a list of supported providers, so you may experience some latency when using these helpers. For most WSGI applications this will not be an issue, but if you'd like to speed it up I suggest fetching the results, storing them in the db or a file, and then pulling from there.

More information can be found in the *ProviderRegistry* API docs.

2.2.4 Parsers

The *micawber.parsers* module contains several handy functions for parsing blocks of text or HTML and either:

- replacing links with rich markup
- extracting links and returning metadata dictionaries

A quick example:

```
import micawber

providers = micawber.bootstrap_basic()

micawber.parse_text('this is a test:\nhttp://www.youtube.com/watch?v=54XHDUOHuzU',␣
↪providers)
```

This will result in the following output:

```
this is a test:
<iframe width="459" height="344" src="http://www.youtube.com/embed/54XHDUOHuzU?fs=1&
↪feature=oembed" frameborder="0" allowfullscreen></iframe>
```

You can also parse HTML using the *parse_html()* function:

```
micawber.parse_html('<p>http://www.youtube.com/watch?v=54XHDUOHuzU</p>', providers)

# yields the following output:
<p><iframe width="459" height="344" src="http://www.youtube.com/embed/54XHDUOHuzU?
↪fs=1&amp;feature=oembed" frameborder="0" allowfullscreen="allowfullscreen"></iframe>
↪</p>
```

If you would rather extract metadata, there are two functions:

- *extract()* (handles text)
- *extract_html()* (handles html)

The *API docs* are extensive, so please refer there for a full list of parameters and functions.

How the parsers determine what to convert

First a couple definitions:

Full representation: A "rich" representation of an embeddable object, for example a flash player or an <img> tag.

Inline representation: A representation of an embeddable object suitable for embedding within a block of text, so as not to disrupt the flow of the text – for example a clickable <a> tag.

There are two parsers that you will probably use the most:

- *parse_text()* for text
  - URLs on their own line are converted into full representations
  - URLs within blocks of text are converted into clickable links
- *parse_html()* for html
  - URLs that are already within <a> tags are passed over
  - URLs on their own in block tags are converted into full representations
  - URLs interspersed with text are converted into clickable links

2.3 Examples

micawber comes with a handful of examples showing usage with

- *django*
- *flask*
- *simple python script*

2.3.1 Django example

The django example is very simple – it illustrates a single view that renders text inputted by the user by piping it through the *oembed()* filter. It also shows the output of the *extract_oembed()* filter which returns a 2-tuple of URL -> metadata. There is also an input where you can experiment with entering HTML.

To run the example:

```
cd examples/django_ex/
./manage.py runserver
```

Check out the example source code.

2.3.2 Flask example

The flask example is almost identical in terms of functionality to the django example. It shows a one-file app with a single view that renders text inputted by the user by piping it through the *oembed()* filter. It also shows the output of the *extract_oembed()* filter which returns a 2-tuple of URL -> metadata. There is also an input where you can experiment with entering HTML.

To run the example:

```
cd examples/flask_ex/
python app.py
```

Check out the example source code.

2.3.3 Python example

The python example is a command-line app that shows the use of the *micawber.providers.ProviderRegistry* and *micawber.providers.bootstrap_embedly*. It runs a loop asking the user to input URLs, outputting rich metadata when possible (view http://embed.ly for a full list of providers).

To run the example:

```
cd examples/python_ex/
python example.py
```

Check out the example source code.

2.4 Flask integration

micawber exposes two Jinja template filters for use in your flask templates:

- *oembed()*
- *extract_oembed()*

You can add them to your jinja environment by using the helper function:

```
from flask import Flask
from micawber.providers import bootstrap_basic
from micawber.contrib.mcflask import add_oembed_filters

app = Flask(__name__)

oembed_providers = bootstrap_basic()
add_oembed_filters(app, oembed_providers)
```

Now you can use the filters in your templates:

```
{% block content %}
  <p>{{ object.body|oembed(html=False, maxwidth=600, maxheight=600) }}</p>
{% endblock %}
```

2.4.1 Flask filter API

The following filters are exposed via the *micawber.contrib.mcflask* module:

micawber.contrib.mcflask.**oembed**(*text*, *urlize_all=True*, *html=False*, ***params*)

Parse the given text, rendering URLs as rich media

Usage within a Jinja2 template:

```
{{ blog_entry.body|oembed(urlize_all=False, maxwidth=600) }}
```

Parameters

- **text** – the text to be parsed, can be HTML
- **urlize_all** – boolean indicating whether to convert bare links to clickable ones
- **html** – boolean indicating whether text is plaintext or markup
- **params** – any additional keyword arguments, e.g. maxwidth or an api key

Return type parsed text with rich content embedded

micawber.contrib.mcflask.**extract_oembed**(*text*, *html=False*, ***params*)

Returns a 2-tuple containing

- a list of all URLs found within the text (if HTML, all URLs that aren't already links)
- a dictionary of URL to metadata provided by the API endpoint

---

Note: Not all URLs listed will have matching entries in the dictionary, since there may not be a provider for them.

Parameters

- **text** – the text to be parsed, can be HTML
- **html** – boolean indicating whether text is plaintext or markup
- **params** – any additional keyword arguments, e.g. maxwidth or an api key

Return type 2-tuple containing a list of *all* urls and a dictionary of url -> metadata

2.4.2 Adding filters to the Jinja Environment

To actually use these filters they must be made available to the application. Use the following function to do this sometime after initializing your `Flask` app:

micawber.contrib.mcflask.**add_oembed_filters**(*app*, *providers*)

Add the `oembed` and `extract_oembed` filters to the jinja environment

Parameters

- **app** – a flask application
- **providers** – a *micawber.providers.ProviderRegistry* instance

Return type (no return value)

2.5 Django integration

First be sure you have added `micawber.contrib.mcdjango` to `INSTALLED_APPS` so that we can use the template filters it defines.

```
# settings.py

INSTALLED_APPS = [
    # ...
    'micawber.contrib.mcdjango',
]
```

micawber provides 4 template filters for converting URLs contained within text or HTML to rich content:

- *oembed()* for plain text
- *oembed_html()* for html
- *extract_oembed()* for extracting url data from plain text
- *extract_oembed_html()* for extracting url data from html

These filters are registered in the micawber_tags library, which can be invoked in your templates:

```
{% load micawber_tags %}

<p>{{ object.body|oembed:"600x600" }}</p>
```

Each filter accepts one argument and one optional argument, due to django's template filters being wack.

Piping a string through the oembed filter (or oembed_html) will convert URLs to things like youtube videos into flash players. A couple things to understand about the parsers:

- the plaintext parser (oembed) will convert URLs *on their own line* into full images/flash-players/etc. URLs that are interspersed within text will simply be converted into clickable links so as not to disrupt the flow of text.
- the HTML parser (oembed_html) will convert URLs that *are not already links* into full images/flash-players/etc. URLs within block elements along with other text will be converted into clickable links as this would likely disrupt the flow of text or produce invalid HTML.

Note: You can control how things are rendered – check out the default templates for reference implementations.

2.5.1 Django filter API

The following filters are exposed via the *micawber.contrib.mcdjango* module:

micawber.contrib.mcdjango.**oembed**(*text*[, *width_height=None*])

Parse the given text, rendering URLs as rich media

Usage within a django template:

```
{{ blog_entry.body|oembed:"600x600" }}
```

Parameters

- **text** – the text to be parsed **do not use HTML**
- **width_height** – string containing maximum for width and optionally height, of format "WIDTHxHEIGHT" or "WIDTH", e.g. "500x500" or "800"

Return type parsed text with rich content embedded

micawber.contrib.mcdjango.**oembed_html**(*html*[, *width_height=None*])

Exactly the same as above except for usage *with html*

Usage within a django template:

```
{{ blog_entry.body|markdown|oembed_html:"600x600" }}
```

micawber.contrib.mcdjango.**extract_oembed**(*text*[, *width_height=None*])
Parse the given text, returning a list of 2-tuples containing url and metadata about the url.

Usage within a django template:

```
{% for url, metadata in blog_entry.body|extract_oembed:"600x600" %}
  <img src="{{ metadata.thumbnail_url }}" />
{% endfor %}
```

Parameters

- **text** – the text to be parsed **do not use HTML**
- **width_height** – string containing maximum for width and optionally height, of format "WIDTHxHEIGHT" or "WIDTH", e.g. "500x500" or "800"

Return type 2-tuples containing the URL and a dictionary of metadata

micawber.contrib.mcdjango.**extract_oembed_html**(*html*[, *width_height=None*])
Exactly the same as above except for usage *with html*

2.5.2 Extending the filters

For simplicity, micawber provides a setting allowing you to create custom template filters. An example use case would be to add a template filter that could embed rich content, but did not automatically "urlize" all links.

Extensions are configured in the `settings` module and take the form of a list of 2-tuples containing:

1. the name for the custom filter
2. a dictionary of keyword arguments to pass in to the `parse` function

```
MICAWBER_TEMPLATE_EXTENSIONS = [
    ('oembed_no_urlize', {'urlize_all': False}),
]
```

Assume this is our template:

```
{% load micawber_tags %}

DEFAULT:
{{ "http://foo.com/ and http://bar.com/"|oembed }}

CUSTOM:
{{ "http://foo.com/ and http://bar.com/"|oembed_no_urlize }}
```

Rendering the above template will produce the following output:

```
DEFAULT:
<a href="http://foo.com/">http://foo.com/</a> and <a href="http://bar.com/">http://
↪bar.com/</a>

CUSTOM:
http://foo.com/ and http://bar.com/
```

Some examples of keyword arguments to override are:

- providers: a *ProviderRegistry* instance

- urlize_all (default `True`): whether to convert *all* URLs to clickable links
- html (default `False`): whether to parse as plaintext or html
- handler: function used to render metadata as markup
- block_handler: function used to render inline links with rich metadata
- text_fn: function to use when parsing text
- html_fn: function to use when parsing html

The magic happens in `micawber.contrib.mcdjango.extension()` – check out the source code for more details.

Note: The `MICAWBER_EXTENSIONS` setting can also be a string path to a module and an attribute containing a similar data structure.

2.5.3 Additional settings

Providers

The most important setting to configure is the module / attribute path to the providers you wish to use. The attribute can either be a ProviderRegistry instance or a callable. The default is:

```
MICAWBER_PROVIDERS = 'micawber.contrib.mcdjango.providers.bootstrap_basic'
```

You can use the bootstrap embedly function, but beware this may take a few seconds to load up:

```
MICAWBER_PROVIDERS = 'micawber.contrib.mcdjango.providers.bootstrap_embedly'
```

If you want to use the embedly endpoints and have an API key, you can specify that in the settings:

```
MICAWBER_EMBEDLY_KEY = 'foo'
```

You can also customize this with your own set of providers. This must be either

- the module path to a *ProviderRegistry* instance
- the module path to a callable which returns a *ProviderRegistry* instance

Here is a quick example showing a custom `ProviderRegistry`:

```
# settings.py
MICAWBER_PROVIDERS = 'my_app.micawber_providers.oembed_providers'
```

```
# my_app/micawber_providers.py
from django.core.cache import cache
from micawber.providers import Provider, bootstrap_basic

oembed_providers = boostrap_basic(cache)

# add a custom provider
oembed_providers.register('http://example.com/\S*', Provider('http://example.com/
↪oembed/'))
```

Default settings for requests

Because of the limitations of django's template filters, we do not have the flexibility to pass in multiple arguments to the filters. Default arguments need to be specified in the settings:

```
MICAWBER_DEFAULT_SETTINGS = {
    'key': 'your-embedly-api-key',
    'maxwidth': 600,
    'maxheight': 600,
}
```

2.5.4 Trying it out in the python shell

```
>>> from django.template import Template, Context
>>> t = Template('{% load micawber_tags %}{{ "http://www.youtube.com/watch?
↪v=mQEWI1cn7HY"|oembed }}')
>>> t.render(Context())
u'<iframe width="480" height="270" src="http://www.youtube.com/embed/mQEWI1cn7HY?fs=1&
↪feature=oembed" frameborder="0" allowfullscreen></iframe>'
```

2.6 API Documentation

2.6.1 Providers

class micawber.providers.**Provider**(*endpoint*, ***kwargs*)

The *Provider* object is responsible for retrieving metadata about a given URL. It implements a method called *request()*, which takes a URL and any parameters, which it sends off to an endpoint. The endpoint should return a JSON dictionary containing metadata about the resource, which is returned to the caller.

Parameters

- **endpoint** – the API endpoint which should return information about requested links
- **kwargs** – any additional url parameters to send to the endpoint on each request, used for providing defaults. An example use-case might be for providing an API key on each request.

request(*url*, ***extra_params*)

Retrieve information about the given url. By default, will make a HTTP GET request to the endpoint. The url will be sent to the endpoint, along with any parameters specified in the `extra_params` and those parameters specified when the class was instantiated.

Will raise a `ProviderException` in the event the URL is not accessible or the API times out.

Parameters

- **url** – URL to retrieve metadata for
- **extra_params** – additional parameters to pass to the endpoint, for example a maxwidth or an API key.

Return type a dictionary of JSON data

class micawber.providers.**ProviderRegistry**([*cache=None*])

A registry for encapsulating a group of *Provider* instances. It has optional caching support.

Handles matching regular expressions to providers. URLs are sent to the registry via its *request()* method, it checks to see if it has a provider that matches the URL, and if so, requests the metadata from the provider instance.

Parameters **cache** – the cache simply needs to implement two methods, `.get(key)` and `.set(key, value)`.

register(*regex*, *provider*)

Register the provider with the following regex.

Example:

```
registry = ProviderRegistry()
registry.register(
    'http://\S*.youtu(\.be|be\.com)/watch\S*',
    Provider('http://www.youtube.com/oembed'),
)
```

Parameters

- **regex** – a regex for matching URLs of a given type
- **provider** – a *Provider* instance

request(*url*, ***extra_params*)

Retrieve information about the given url if it matches a regex in the instance's registry. If no provider matches the URL, a `ProviderException` is thrown, otherwise the URL and parameters are dispatched to the matching provider's *Provider.request()* method.

If a cache was specified, the resulting metadata will be cached.

Parameters

- **url** – URL to retrieve metadata for
- **extra_params** – additional parameters to pass to the endpoint, for example a maxwidth or an API key.

Return type a dictionary of JSON data

micawber.providers.**bootstrap_basic**(*[cache=None[, registry=None]]*)

Create a *ProviderRegistry* and register some basic providers, including youtube, flickr, vimeo.

Parameters

- **cache** – an object that implements simple `get` and `set`
- **registry** – a `ProviderRegistry` instance, which will be updated with the list of supported providers. If not specified, an empty `ProviderRegistry` will be used.

Return type a `ProviderRegistry` with a handful of providers registered

micawber.providers.**bootstrap_embedly**(*[cache=None[, registry=None[, **kwargs]]]*)

Create a *ProviderRegistry* and register as many providers as are supported by embed.ly. Valid services are fetched from http://api.embed.ly/1/services/python and parsed then registered.

Parameters

- **cache** – an object that implements simple `get` and `set`
- **registry** – a `ProviderRegistry` instance, which will be updated with the list of supported providers. If not specified, an empty `ProviderRegistry` will be used.

- **kwargs** – any default keyword arguments to use with providers, useful for specifying your API key

Return type a ProviderRegistry with support for embed.ly

```
# if you have an API key, you can specify that here
pr = bootstrap_embedly(key='my-embedly-key')
pr.request('http://www.youtube.com/watch?v=54XHDUOHuzU')
```

micawber.providers.**bootstrap_noembed**(*[cache=None[, registry=None[, **kwargs]]]*)

Create a *ProviderRegistry* and register as many providers as are supported by noembed.com. Valid services are fetched from http://noembed.com/providers and parsed then registered.

Parameters

- **cache** – an object that implements simple `get` and `set`
- **registry** – a `ProviderRegistry` instance, which will be updated with the list of supported providers. If not specified, an empty `ProviderRegistry` will be used.
- **kwargs** – any default keyword arguments to use with providers, useful for passing the `nowrap` option to noembed.

Return type a ProviderRegistry with support for noembed

```
# if you have an API key, you can specify that here
pr = bootstrap_noembed(nowrap=1)
pr.request('http://www.youtube.com/watch?v=54XHDUOHuzU')
```

micawber.providers.**bootstrap_oembedio**(*[cache=None[, registry=None[, **kwargs]]]*)

Create a *ProviderRegistry* and register as many providers as are supported by oembed.io. Valid services are fetched from http://oembed.io/providers and parsed then registered.

Parameters

- **cache** – an object that implements simple `get` and `set`
- **registry** – a `ProviderRegistry` instance, which will be updated with the list of supported providers. If not specified, an empty `ProviderRegistry` will be used.
- **kwargs** – any default keyword arguments to use with providers

Return type a ProviderRegistry with support for noembed

2.6.2 Parsers

Functions for parsing text and HTML

micawber.parsers.**parse_text_full**(*text*, *providers*[, *urlize_all=True*[, *handler=full_handler*[, ***params*]]])

Parse a block of text, converting *all* links by passing them to the given handler. Links contained within a block of text (i.e. not on their own line) will be handled as well.

Example input and output:

```
IN: 'this is a pic http://example.com/some-pic/'
OUT: 'this is a pic <a href="http://example.com/some-pic/"><img src="http://
↪example.com/media/some-pic.jpg" /></a>'
```

Parameters

- **text** – a string to parse
- **providers** – a ProviderRegistry instance
- **urlize_all** – whether to convert all urls irrespective of whether a provider exists
- **handler** – function to use to convert metadata back into a string representation
- **params** – any additional parameters to use when requesting metadata, i.e. a maxwidth or maxheight.

micawber.parsers.**parse_text**(*text*, *providers*[, *urlize_all=True*[, *handler=full_handler*[, *block_handler=inline_handler*[, ***params*]]]])

Very similar to the above *parse_text_full()* except URLs *on their own line* are rendered using the given handler, whereas URLs within blocks of text are passed to the block_handler. The default behavior renders full content for URLs on their own line (e.g. a flash player), whereas URLs within text are rendered simply as links so as not to disrupt the flow of text.

Parameters

- **text** – a string to parse
- **providers** – a ProviderRegistry instance
- **urlize_all** – whether to convert all urls irrespective of whether a provider exists
- **handler** – function to use to convert links found on their own line
- **block_handler** – function to use to convert links found within blocks of text
- **params** – any additional parameters to use when requesting metadata, i.e. a maxwidth or maxheight.

micawber.parsers.**parse_html**(*html*, *providers*[, *urlize_all=True*[, *handler=full_handler*[, *block_handler=inline_handler*[, ***params*]]]])

Parse HTML intelligently, rendering items on their own within block elements as full content (e.g. a flash player), whereas URLs within text are passed to the block_handler which by default will render a simple link. Also worth noting is that URLs that are already enclosed within a <a> tag are skipped over.

Note: requires BeautifulSoup or beautifulsoup4

Parameters

- **html** – a string of HTML to parse
- **providers** – a ProviderRegistry instance
- **urlize_all** – whether to convert all urls irrespective of whether a provider exists
- **handler** – function to use to convert links found on their own within a block element
- **block_handler** – function to use to convert links found within blocks of text
- **params** – any additional parameters to use when requesting metadata, i.e. a maxwidth or maxheight.

Functions for extracting rich content from text and HTML

micawber.parsers.**extract**(*text*, *providers*, ***params*)

Extract all URLs from a block of text, and additionally get any metadata for URLs we have providers for.

Parameters

- **text** – a string to parse
- **providers** – a `ProviderRegistry` instance
- **params** – any additional parameters to use when requesting metadata, i.e. a maxwidth or maxheight.

Return type returns a 2-tuple containing a list of all URLs and a dictionary keyed by URL containing any metadata. If a provider was not found for a URL it is not listed in the dictionary.

`micawber.parsers.`**extract_html**(*html*, *providers*, ***params*)

Extract all URLs from an HTML string, and additionally get any metadata for URLs we have providers for. Same as *extract()* but for HTML.

Note: URLs within <a> tags will not be included.

Parameters

- **html** – a string to parse
- **providers** – a `ProviderRegistry` instance
- **params** – any additional parameters to use when requesting metadata, i.e. a maxwidth or maxheight.

Return type returns a 2-tuple containing a list of all URLs and a dictionary keyed by URL containing any metadata. If a provider was not found for a URL it is not listed in the dictionary.

2.6.3 Cache

class `micawber.cache.`**Cache**

A reference implementation for the cache interface used by the `ProviderRegistry`.

get(*key*)

Retrieve the key from the cache or `None` if not present

set(*key*, *value*)

Set the cache key `key` to the given `value`.

class `micawber.cache.`**PickleCache**([*filename='cache.db'*])

A cache that uses pickle to store data.

Note: To use this cache class be sure to call *load()* when initializing your cache and *save()* before your app terminates to persist cached data.

---

load()

Load the pickled data into memory

save()

Store the internal cache to an external file

class `micawber.cache.`**RedisCache**([*namespace='micawber'*[, ***conn*]])

A cache that uses Redis to store data

Note: requires the redis-py library, `pip install redis`

Parameters

- **namespace** – prefix for cache keys
- **conn** – keyword arguments to pass when initializing redis connection

2.7 Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index