WebResourceManager

Table of contents

Introduction

WebResourceManager is a class for helping manage a database of web resources. WebResourceManager creates a UUID (Universally Unique Identifier) for the web resource, saves the information in a JSON (labeled < UUID >.json), and builds maintain a dictionary from URL to UUID. Using this uniform data storage system and a simple API, WebResourceManager makes storing and querying the contents and source files (such as HTML and PDF) of web resources much simpler.

Documentation

`__init__(rel_path=None)`

- **rel_path** is the relative path from the current working directory to the directory where the files you'd like to work with are.

Instantiates the object and makes a new instance of the **Classifier**.

`__iter__()`

Returns an iterator for the JSONs holding the web resource information.

`__getitem__(key)` **and get(key)**

- **key** is the UUID of the web resource content you would like to access.

Prints an error message containing the name of the file if the file can't be opened.

`__len__()`

Returns the number of web resources currently being tracked by the WebResourceManager object.

`__repr__()` and `__str__()`

Returns a string representation of the WebResourceManager object using JSON.

absorb_file_manager(other_file_manager)

- **other_file_manager** is another WebResourceManager object.

This object starts tracking all of the files in the **other_file_manager** by adding the files to its dictionary. **other_file_manager** and its files are not altered in any way.

get_corpus(process_all=False)

- **process_all** is a parameter that allows you specify whether or not you would like to reprocess a document which already has a **corpus** field. If the document already has a **corpus** field and **process_all** == False, the document is not processed and the field is added to the corpora. If the document already has a **corpus** field and **process_all** == True, the document is reprocessed and its **corpus** field is overwritten using the new output.

Specifically, it this function decodes using the "UTF-8" codec, ignoring errors, uses regular expressions to remove email addresses, special characters and numbers, then removes all words of length < 3 from the corpus and stems using NLTK's PorterStemmer.

Note: This method serves the function of the now deprecated corpusBuilder's `__init__()` and **filter_dict()** functions.

get_relevance_score(document)

- **document** is a string, generally the text attribute of a document.

Returns the relevance score using **classifier**.

get_TaggedDocuments()

A generator that yields the **corpus** fields of the contained documents as gensim's TaggedDocument objects with the **id** and **query** fields as tags.

get_texts()

Simple generator function so you can iterator over the text fields of documents. Can be used as a shortcut for iterating over the documents and then extracting the text field. See Usage for examples.

get_uuid(url)

- **url** is the URL of the web resource

Returns the UUID of the web resource with the specified URL.

load(file_name=None)

- **file_name** is the name of the file you wish to load a WebResourceManager from.

Reads in the WebResourceManager object from a file. The default is "data/webresourcemanager.json" and if rel_path was specified it will be "rel_path/data/webresourcemanager.json"

read_in_from_directory(directory)

- **directory** is the name of the directory you would like to read files in from.

NOTE: In order to avoid weird interactions, this function ignores **rel_path**. You put the path from the current working directory in as **directory**.

For each non-directory file in the directory, it attempts to load the file and add it to its dictionary. If this fails, it prints an error message that reads:

> "File {} was not in the proper format to be tracked with WebResourceManager".format(file)

read_in_from_iterator(iterator_of_docs)

- **iterator_of_docs** is an iterator of documents which are just dictionaries with a **text**, **query**, **url**, and either **html** or **pdf** attributes.

For each document, the method generates a UUID, adds the URL to UUID mapping to the dictionary, writes the appropriate source file (< UUID >.html or < UUID >.pdf), addes a timestamp (seconds since epoch), and writes a content JSON with the **id**, **query**, **text**, **time**, and **url** fields.

rank_by_relevance()

For each file in the WebResourceManager, it passes it's text attribute to **get_relevance_score** and assigns the return value to the document's **relevance_score** attribute.

string_to_uuid(string)

- **string** is a string, in practice it is generally the URL of the web resource

Returns the UUID for the web resource.

save(file_name=None)

- **file_name** is the name of the file you wish to save the WebResourceManager to.

Writes the WebResourceManager object to the **file_name** if specified. Otherwise, it writes to "data/webresourcemanager.json" or "rel_path/data/webresourcemanager.json" if rel_path was specified.

train_classifier()

Trains the **classifier** on the WebResourceManager's documents

Basic Usage

This section acts as a tutorial for how to use WebResourceManager without reading through every function in the Documentation section.

What is a WebResourceManager?

A WebResourceManager is basically a dictionary that converts URLs to UUIDs (to label and lookup files) with some fun functionalities. For example, because of the file labeling and data storage scheme, given a UUID or URL of a web resource, we can instantly pull up the source file (HTML or PDF) or the document file (attributes such as the text and URL).

How do I Create a WebResourceManager?

You create a WebResourceManager object just like any other object, and the constructor doesn't do any heavy lifting.

```
wrm = WebResourceManager()
```

The only thing interesing in the constructor is the **rel_path** keyword argument which allows you set the relative path from your current working environment to the directory where your data is housed. For example, let's say your directory layout looks like this:

```
project\
  |----data\
  |      |---docs\
  |      |---source\
  |----scripts\
         |----my_file.py
```

To use WebResourceManager from **my_file.py** (for example with an import statement), you want to set the **rel_path** variable to ".." because WebResourceManager always expects to have access to a "data" directory as well as "data/docs" and "data/source" subdirectories. However, you could also just put a "data" fold in "scripts" with both "docs" and "source" subdirectories as well.

How do I Populate a WebResourceManager?

To populate a WebResourceManager, you can use **load()** if you already have written a WebResourceManager object to a JSON. The **load()** function looks to "data/webresourcemanager.json" by default, but if you specify **rel_path** will look to "rel_path/data/webresourcemanager.json", and you can also just specify your own file_name which ignores **rel_path** and the "webresourcemanager.json" convention.

If you haven't written out a WebResourceManager or want to add more documents, you have two options: **read_in_from_iterator()** and **read_in_from_directory()**.

read_in_from_iterator() takes an iterator of documents. Documents are simply dictionaries with a **text**, **url** and either **html** or **pdf** attribute. The function generates a UUID using **string_to_uuid(string)**, puts the ID and URL pair in the dictionary, adds a time stamp, writes the source file, and writes the document file.

read_in_from_directory() requires documents that have already been written by a WebResourceManager and when you specify the a directory (ignoring **rel_path**), it attempts to read in all of the document files in the directory, printing error messages for files it cannot parse.

```
tmp = WebResourceManager() \
tmp.load(os.path.join("data/webresourcemanager", file_name)
```

If you have another WebResourceManager, you can also "absorb" those files by adding them to your dictionary. This is different from copying because you retain any files you were tracking previously. It is also important to note that the physical files and the other WebResourceManager are not altered during this function call, you just gain their files.

```
wrm = WebResourceManager() \
tmp = WebResourceManager() \
wrm.absorb_file_manager(tmp)
```

Accessing Files in WebResourceManager

Once you have a WebResourceManager object and with files, accessing the files is very easy!

To get the UUID of a web resource if you have its URL just use **get_uuid()**

```
wrm.get_uuid("http://money.cnn.com/quote/forecast/forecast.html?symb=BASFY") \
--> bd841c9a-ea85-54e4-9efa-1d627abbfa36
```

To access a file once you have a UUID, you can use **get()** of just use square brackets [ ] to access the document file.

```
wrm["bd841c9a-ea85-54e4-9efa-1d627abbfa36"]
```

To access a specific attribute of the file, because document files are just dictionaries, you can just append [attribute] to the end of that call. For example, if you wanted the query that generated the document, you would use:

```
wrm["bd841c9a-ea85-54e4-9efa-1d627abbfa36"]['query'] \
--> test
```

Iterating over files is a very common task and WebResourceManager has made that easy as well! You have a few options. First, you could iterate over WebResourceManager's **url_to_uuid** dictionary by calling **values()** on it to get a list of UUIDs in the file.

```
for elem in wrm.url_to_uuid.values(): \
    print(elem)
```

This is not recommend and there is built in support for iterating! The `__iter__()` function is a generator which returns an iterator over the files. This means that when you make a "for x in WebResourceManager" statement, it will give you the files you want. The following example prints the **id** attribute of the documents in the WebResourceManager:

```
for elem in wrm: \
    print(elem['id'])
```

We are of couse working with text documents however, so to save you the trouble of iterating and explicitly saying ['text'], there is the **get_texts()** generator which returns an iterator over the **text** attributes of the tracked files.

```
for elem in wrm.get_texts(): \
    print(elem)
```