Pyalveo Documentation

Release 1.1

Steve Cassidy

Dec 19, 2018

Contents

Contents:

CHAPTER 1

Installation

At the command line:

```
$ easy_install pyalveo
```

Or, using pip:

```
$ pip install pyalveo
```

CHAPTER 2

Usage

This module provides an interface to the Alveo API which supports a wide variety of operations on data stored in Alveo.

Access to Alveo requires authentication and this is mediated in the API by an API Key. The key can be obtained from the Alveo website and must be made available to any scripts using the *pyalveo* module.

All access to the API is carried out through a *client* instance. Methods defined on this instance correspond to the various API methods. The simplest way to create a client is to provide the API URL and API Key as arguments:

```
import pyalveo

client = pyalveo.Client(api_key='xxxx',
                        api_url='https://app.alveo.edu.au/')
```

Note that the API URL is not hard-coded into the module so needs to be provided. This will normally be the default Alveo URL but could also reference the staging server for testing (in particular, if you want to test upload scripts the staging server is available - please contact the admin team).

The disadvantage of this method is that the API Key is included in the code meaning that it can't be published without removing it. As an alternative, a user can download a file *alveo.config* from Alveo (see instructions on the page linked above) and the configuration parameters can be read from this file. The contents of this file (formatted for readability) are:

```
{
 "base_url":"https://app.alveo.edu.au/",
 "apiKey":"wCTRNdjyxxyzzymp9B",
 "cacheDir":"wrassp_cache"
 }
```

(the last entry in the file is the directory that will be used for caching if configured). The client can now be created by passing the name of this configuration file:

```
client = pyalveo.Client(configfile="alveo.config")
```

If no config file is specified then the module will search for a file called *alveo.config* in the user's home directory and read the settings from there. This is the best option for most user-facing scripts. If users are instructed to save this file in their home directory, then any script or application that uses the pyalveo module will find the configuration file there.

2.1 Caching

The module is able to cache responses from the API for faster access on repeated runs of a script. If the *use_cache* option is *True* (the default) then any data or metadata downloaded from the API will be cached in a local directory. If the same request is made again, the cached version will be used resulting in a faster response, particularly when dealing with larger data files. Note that this can result in significant disk usage if lots of data is accessed so use this with caution. The cache can be disabled when the client is created:

```
client = pyalveo.Client(use_cache=False)
```

2.2 Collections

Using the API, a script can access all data and metadata that a user has permission to see. Data on Alveo is organised into *collections*. To get a list of collections:

```
collections = client.get_collections()
```

This returns a list of (collection_name, collection_uri) tuples for all collections accessible to the user.

2.3 Items and Documents

An item is a container for one or more documents (files) representing a single linguistic event, usually a single text or one or more audio/video recordings with associated annotations. Each item has a URI that can be used to access metadata about the item and the documents that are attached to it.

```
item_uri = "https://app.alveo.edu.au/catalog/cooee/4-214"
item = client.get_item(item_uri)
```

The return value of *get_item* is an instance of `pyalveo.objects.Item` which has methods for retrieving metadata and documents:

```
itemid = item.metadata()['identifier]
docs = item.get_documents()
```

the last call returns a list of Document instances (`pyalveo.objects.Document`) that can be used to access the document and it's metadata. For example, to download all attached files to a local directory *data*:

```
for doc in item.get_documents():
    doc.download_content('data')
```

Alternatively you can get the contents of the document:

```
doc = item.get_document(0)
content = doc.get_content()
```

Some items have a property *primary_text* which is the plain text representation of the item - usually this is the case if the item contains a simple text document. In this case you can use the *get_primary_text* method of the item to retrieve the content:

```
text = item.get_primary_text()
```

2.4 Item Lists

An item list is a list of items created by a user. The intention is that an item list contains a set of items to be used in a experiment so that repeating the analysis is easy. Item lists have a URL and can have basic metadata associated with them. They can be private to an individual or shared with others. If you are running an experiment with data from Alveo, it is a good idea to use an Item list as the input to your script. Then the same workflow can be used on different item lists to repeat the analysis. Similarly, if you script queries a collection for certain data, you could store the result as an item list for future use.

```
lists = client.get_item_lists()
```

Returns a dictionary with two keys *own* and *shared*. Each key contains a list of item lists belonging to the user (*own*) or shared globally. Each item list entry is a dictionary:

```
{
   'name': 'mava s1-s20',
   'item_list_url': 'https://app.alveo.edu.au/item_lists/1090',
   'num_items': 20,
   'shared': True
}
```

Retrieving an individual item list returns an instance of `pyalveo.objects.ItemList` which can be used to access the list of items. The following example downloads the primary text of every item in a list:

```
texts = []
for item in client.get_item_list('https://app.alveo.edu.au/item_lists/53'):
    texts.append(item.get_primary_text())
```

Similarly we could download some or all documents attached to an item. This example downloads just the *.wav* files:

```
texts = []
for item in client.get_item_list('https://app.alveo.edu.au/item_lists/53'):
    for doc in item.get_documents():
        if doc.get_filename().endswith('.wav'):
            doc.download_content('data')
```

2.5 Speakers

Speakers are associated with collections and items. A speaker is a collection of metadata describing a speaker (or author of a text) with an identifier that is unique within a collection. Speaker metadata is available from a URL and that URL can be associated with an item.

```
speakers = client.get_speakers('austalk')
meta = client.get_speaker(speakers[0])
```

The first call returns a list of speaker URLs associated with the collection. *get_speaker* takes one of these URLs and returns the metadata description for that speaker as a dictionary. You can add new speakers with *add_speaker* and delete them with *delete_speaker*. Speaker URLs appear in the metadata of items, eg. in the *olac:speaker* property.

2.6 Contributions

Contributions are collections of documents uploaded by users and associated with items in a particular collection. For example, if I download some audio data from the Austalk collection and transcribe it to generate a text file for each item, I can share the resulting files by creating a *contribution*. Each file in the contribution will be associated with the original item (usually by matching filenames). Other users can then download these files as a zip file via the web interface. You can also work with them via the API.

To find out what contributions are available:

```
client.get_contributions()
```

This returns a dictionary with keys *own* and *shared* contianing lists of contributions, each entry in the list is a dictionary:

```
{'id': 6,
'name': 'Austalk Manual Transcriptions',
'url': 'https://app.alveo.edu.au/contrib/6'
}
```

A contribution can be created via the API. Contributions are associated with a particular collection and can have associated metadata.

```
meta = {
    "contribution_name": "HelloWorld",
    "contribution_collection": "demotext",
    "contribution_text": "This is contribution description",
    "contribution_abstract": "This is contribution abstract"
}
result = client.create_contribution(meta)
```

The return value is a dictionary containing details of the contribution including the metadata:

```
{'description': 'This is contribution description',
 'documents': [],
 'id': '29',
 'metadata': {'abstract': '"This is contribution abstract"',
              'collection': 'https://app.alveo.edu.au/catalog/demotext',
              'created': '2018-12-06T05:46:11Z',
              'creator': 'Data Owner',
              'title': 'HelloWorld'},
 'name': 'HelloWorld',
 'url': 'https://app.alveo.edu.au/contrib/29'}
```

To add documents to the contribution we use the normal *add_document* method but add the *contrib_id* argument to indicate that it should be associated with this contribution. Normally, only the owner of a collection is allowed to add documents to an item but if the document is associated with a collection, any user can upload documents.

```
itemurl = 'https://app.alveo.edu.au/catalog/demotext/2006-05-28-19'
result = client.add_document(itemurl, 'testfile2.txt', metadata={}, content='hello␣
↪world', contrib_id=result['id'])
```

The script needs to know which item the new document should be associated with, this might be done with reference to an item list that was used to download the original data.

CHAPTER 3

Examples

This page contains some example scripts written using the API.

3.1 Download Austalk Data

This script downloads files from the Austalk collection based on an item list. It could be adapted to work with any collection. It reads the metadata for the item list and iterates over the items it contains. For each item it downloads files ending in *speaker16.wav* and *TextGrid*. Files are stored in directories named for the speaker identifier for the item.

Note that in creating the client the cache is disabled to prevent versions of downloaded files being cached.

```
from __future__ import print_function
import os
import pyalveo

item_list_name = 'over-50-hvd'

# directory to write downloaded data into
outputdir = "data"

client = pyalveo.Client(use_cache=False)

itemlist = client.get_item_list_by_name(item_list_name)

if not os.path.exists(outputdir):
    os.makedirs(outputdir)

for itemurl in itemlist:
    item = client.get_item(itemurl)
    meta = item.metadata()

    speakerurl = meta['alveo:metadata']['olac:speaker']
    speaker_meta = client.get_speaker(speakerurl)
```

(continues on next page)

(continued from previous page)

```
    speakerid = speaker_meta['dcterms:identifier']

    # write out to a subdirectory based on the speaker identifier
    subdir = os.path.join(outputdir, speakerid)
    if not os.path.exists(subdir):
        os.makedirs(subdir)

    for doc in item.get_documents():
        filename = doc.get_filename()

        if filename.endswith('speaker16.wav') or filename.endswith('.TextGrid'):
            print(filename)
            doc.download_content(dir_path=subdir)
```

CHAPTER 4

Credits

- Steve Cassidy <Steve.Cassidy@mq.edu.au>
- Matt Atcheson
- Michael Bauer
- David Irving
- Karl Li

CHAPTER 5

Indices and tables

- genindex
- modindex
- search