2-17-2020

Scraping BePress: Downloading Dissertations for Preservation

Stephen Zweibel
The Graduate Center, City University of New York

code{4}lib JOURNAL

Issue 47, 2020-02-17 ISSN 1940-5758

Scraping BePress: Downloading Dissertations for Preservation

This article will describe our process developing a script to automate downloading of documents and secondary materials from our library's BePress repository. Our objective was to collect the full archive of dissertations and associated files from our repository into a local disk for potential future applications and to build out a preservation system.

Unlike at some institutions, our students submit directly into BePress, so we did not have a separate repository of the files; and the backup of BePress content that we had access to was not in an ideal format (for example, it included "withdrawn" items and did not effectively isolate electronic theses and dissertations). Perhaps more importantly, the fact that BePress was not SWORD-enabled and lacked a robust API or batch export option meant that we needed to develop a data-scraping approach that would allow us to both extract files and have metadata fields populated. Using a CSV of all of our records provided by BePress, we wrote a script to loop through those records and download their documents, placing them in directories according to a local schema. We dealt with over 3,000 records and about three times that many items, and now have an established process for retrieving our files from BePress. Details of our experience and code are included.

by Stephen Zweibel

Institutional Context

We at the GC share a BePress Digital Commons instance with the rest of the CUNY Libraries. We work entirely with Doctoral and Master's students, who often have academic work that they submit to our Digital Commons, which we call CUNY Academic Works. In fact, in order to graduate, students generally must submit their final thesis/capstone/dissertation to Academic Works. Thus all or nearly all dissertations and theses published since we began working with BePress are on Academic Works.

With corporate changes, including the acquisition of BePress by Elsevier in 2018, we in the library began to feel like we were losing control of our students' work. Additionally, it became clear to us that Academic Works is not a complete archival solution, because it

does not provide archival quality PDFs. In response, we decided to explore creating our own digital archive using Archivematica.

In order to accomplish this, we needed the documents and metadata that were stored in Academic Works. When we requested our data from BePress, it was supplied, but we found its format and organization to be unideal, and a difficult starting point for our work. For example, it included “withdrawn” items and did not effectively isolate electronic theses and dissertations. Working with this dataset would have required a great deal of manual work confirming records and editing metadata, far too much for a team of three librarians exploring this in their quieter moments.

So, instead, I began to look for programmatic methods to get our documents and metadata out of Academic Works ourselves. The most likely solution would have been an API (Application Programming Interface), or so I thought. Often, websites with a great number of documents, especially those with an academic focus, have an API that allows for programmatic searching and harvesting of records. BePress does indeed provide an API (https://www.bepress.com/reference_guide_dc/digital-commons-oai-harvesting/), but there is an important limitation. Supplemental files are not exposed through BePress’s OAI (Open Archives Initiative) interface, so these files, which may include a database, set of images, or anything not part of the primary document, would be invisible to any simple harvester I might write. This would render any result incomplete, and thus not at all an archival copy.

On top of that, BePress does not provide a SWORD (Simple Web-service Offering Repository Deposit) API, which would have made downloading our documents a relatively simple task.

Problem

The problem became a technical one, revolving around the difficulty of retrieving each student’s thesis/dissertation/capstone as well as all supplemental files they had uploaded to Academic Works. As far as we could tell, there was no method by which BePress had exposed the locations or existence of supplemental files. And yet, there they were, listed on the index page of each document of Academic Works! So close and yet so far.

Our final aims were:

1. to download every thesis/dissertation/capstone in Academic Works,
2. to match each one with its corresponding metadata,

3. and to relocate those files and metadata to a set of directories with unique IDs organized such that they could be imported into Archivematica.

At the moment I was stuck at step one, without a reliable method of getting the files we wanted (and only those files) out of the BePress repository. Since the most likely solution to our troubles (a robust API) was not forthcoming, and asking nicely didn't get us anywhere, the next method to turn to would be web scraping.

Scraping

Web scraping is often a last resort in situations where it is necessary to get data off of a website. When describing the technique, I often say that it is like creating a program to click through a website for you using an html map you've written for the program. This technique can be unreliable, in that when websites update their architecture, or even when they make small changes to their html, the web scraping script will break, since the map you've provided no longer applies.

To build the "map" required for the web scraping script, I needed to know the URL locations of each document we wanted to download. Hence the problem: up to this point we had the URL of the primary documents, but none of the supplementary files.


DH Box: A Virtual Computer Lab in the Cloud

Stephen Zweibel, Graduate Center, City University of New York

Follow

Download

INCLUDED IN

Digital Humanities Commons

SHARE

Date of Degree
2-2016

Document Type
Capstone Project

Degree Name
M.A.

Program
Liberal Studies

Advisor
Matthew Gold

Subject Categories
Digital Humanities

Keywords
Infrastructure, Installation, Workshops, Pedagogy, Omeka, Programming

Abstract
Education in the use and manipulation of Digital Humanities (DH) tools is rife with challenges, stemming from issues of knowledge (ie. how to install, configure, and work with DH tools on a variety of devices) and resources (access to hardware and software). Particularly in an academic environment, humanities scholars may not only lack access to institutional devices, but their drive to experiment with new programs may run counter to the security concerns of their information technology department personnel. DH Box, originally developed as a project for an introduction to DH course, was conceived as a solution to these challenges: a toolset that would offer simpler setup and execution of DH instruction, to create an easy-to-deploy learning environment for DH students. The project has cycled through a number of iterations, from local installs to a cloud-based solution, and in its current form has been tested in several classroom and workshop scenarios. This testing has confirmed its utility with respect to the challenges of learning DH tools and has informed its continued development. With growing awareness and further development, it is anticipated that DH Box will see still wider adoption.

Recommended Citation
Zweibel, Stephen, "DH Box: A Virtual Computer Lab in the Cloud" (2016). CUNY Academic Works.
https://academicworks.cuny.edu/gc_etds/787

Additional Files

dhbox-master.zip (1270 kB)
DH Box Git Repository

Figure 1.

However, because I had resorted to scraping the pages, and each “index page”, as they are called, lists the supplemental files associated with the document (and, crucially, provides a link to download that file!), I was able to find everything according to a predictable schema and place each file in our predetermined file structure. The rest of this article will demonstrate the process to scrape a website according to a provided schema, or map.

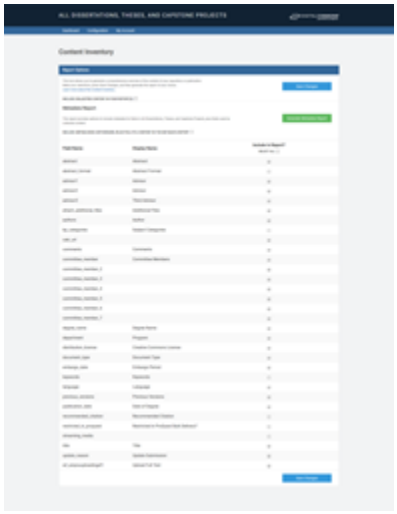

Figure 2.

The Schema

Luckily, BePress provides a “Content Inventory” report (see image) that allowed us to just pull the metadata of the collections we were interested in, in this case all dissertations/theses, etc.. This would allow us to complete our second objective, pairing metadata to files. More importantly, for each row (which corresponded to a record), there was a corresponding URL for the primary document’s index page (see image), where, again, each supplementary file is listed. At this point,I knew what was required: to write a script that would proceed line by line down the Content Inventory report, go to each index page, and download the primary document and any supplementary files found there.

To start with, I needed my script to read the ‘Content Inventory’ CSV, and turn each row into an object, like:

```
1 import csv
2 CSV = open('bepress.csv')
3 csv_reader = csv.DictReader(CSV)
```

Then, looping through the CSV:

```
all_works = []
for item in csv_reader:
    author = item['author']
    pdf = item['pdf_filename']
    …
    together = {'filename': pdf, 'author': author, …}
    all_works.append(together)
```

And so on. Now I have a list of dictionaries that contain the metadata of each item in our collection. As I do this, I can transform each string of text into my desired format. For instance, an author's name could go from "Jane Smith" to "Smith, Jane".

Requests

When I interact with websites, I turn to the Requests library. It's a library for Python that simplifies getting the data from a website into a format that I can play around with. For instance:

```
import requests
r = requests.get('https://academicworks.cuny.edu')
website_text = r.text
```

The first line would get me the front page of Academic Works, and 'r.text' would contain the entire page as a string, ready for me to parse. This would work equally well for calling a web API–better, in fact, as that would be much simpler to parse.

I wrote a function to get the files from Academic Works, simplified below:

```
def fetch_url(entry):

# define the location where I want the files to go
    path = "dissertations/"+ entry['last_name'] + "_" +
str(entry['unique_id'])

# the file URL I got from the csv
    uri = entry['download_url']
    os.mkdir(path)
    r = requests.get(uri, stream=True)

# checking if the URL works, and then starting the download
```

```
    if r.status_code == 200:
        with open(path + '/' + entry['filename'], 'wb') as f:
            for chunk in r:
                f.write(chunk)

# getting the supplemental files, discussed below
       for index, part in enumerate(entry['supplementals']):
            get_supplementals(index, part, entry['article_number'][0],
path)
    return path
```

Here we accomplish our third objective: to relocate our files and metadata to a set of directories with unique IDs.

The trick to getting the supplemental files is the only real web scraping necessary in this approach. As it turns out, the format BePress uses to create the URLs for supplementary files is quite simple, provided you know how to read it. Here is one below:

https://academicworks.cuny.edu/cgi/viewcontent.cgi?filename=0&article=1784&context=gc_etds&type=additional

The first part of that url should be familiar to any user of the web: https://academicworks.cuny.edu. We are interested in the part that comes after the question mark: those are known as **query parameters**.

filename=0&article=1784&context=gc_etds&type=additional

The query parameters follow the ? in the request, and are separated from one another by the & symbol. To determine what each parameter does, we can only experiment, because documentation is not available to us. The first query parameter, filename=0, gets the first supplementary file (therefore 'filename=1' would get the second!). The second parameter, article=1784, refers to the unique IDof the primary document, which we know from our CSV. 'Context' is the particular collection we are looking in, and we don't have any use for 'Type'!

So, a hypothetical 5th supplementary file would have the URL of:

https://academicworks.cuny.edu/cgi/viewcontent.cgi?filename=**4**&article=1784&context=gc_etds&type=additional

This sort of URL is sometimes called an "internal API", inasmuch as it is an undocumented API; we can still use it to get what we want.

The function is as follows:

```
def get_supplementals(which_one, filename, article_number, path):
    download_url = 'https://academicworks.cuny.edu/cgi/viewcontent.cgi' \
                   + '?filename=' + str(which_one) \
                   + '&article=' + str(article_number) \
                   + '&context=gc_etds' \
                   + '&type=additional'

# Download the file
    s_file = requests.get(download_url)
    if "ERROR: No such file is available for this article" in
s_file.text:
        return
    with open(path + '/' + filename, 'wb') as f:
        f.write(s_file.content)
```

The variable ‘download_url’ is constructed for each entry out of the filename retrieved from the Content Inventory CSV. The ‘article_number’ parameter is which supplementary file (1, 2, 3, etc.) as discussed above.

In all of our metadata, we still don’t know how many supplementary files a document has, or whether a document even has any supplementary files! So, we loop through and try to download the first supplementary file, the second, and so on, until we get an error message, which in this case comes as a warning from BePress: “ERROR: No such file is available for this article”. When we see this text, we stop.

Result

Now we have achieved our goals:

1. to download every thesis/dissertation/capstone in Academic Works, we used the Requests library and BePress’s internal API;
2. to match each one with its corresponding metadata, we used the “Content Inventory” provided by BePress;
3. and to relocate those files and metadata to a set of directories with unique IDs organized in order to be imported into Archivematica, we wrote a line in Python that created and named a directory according to each document’s unique ID.

All this has allowed us to continue moving forward on our project to make a real archival backup of our dissertations, theses, and capstones. Of course, this method has

limitations, and a few potential improvements come to mind, for example, a way of ensuring that the downloaded files are complete, not corrupted or mistaken, for instance hash checking. (The way I accomplished this, which is not a complete solution, was to download the whole set multiple times and comparing the sets.)