Automating Digital Collection Processes

Christopher Starcher
Robert Luttrell

We Have So Much Time And So Little To See

There is a lot of content in this presentation. I have shortened it as much as I could without omitting important information.

I will try to get through it as quickly as I can, and I will be glad to stay as late as needed to answer any questions.

My Journey

It seemed to me that the best way to structure this presentation so that it made the most amount of sense was to do so chronologically.

Many Moons Ago

Many moons ago, we developed a dark archive for digital collections. To standardize the content of the archive, we came up with this collection structure.

Archive Structure

ttu_ices01

content

archive

ttu_ices01_000001_000001.tif

display

ttu_ices01_000001.pdf

We standardized the folder names and file names based on collection codes that are assigned for each new collection.

Fortunately, this standardized structure lends itself well to automation.

Necessity And Invention

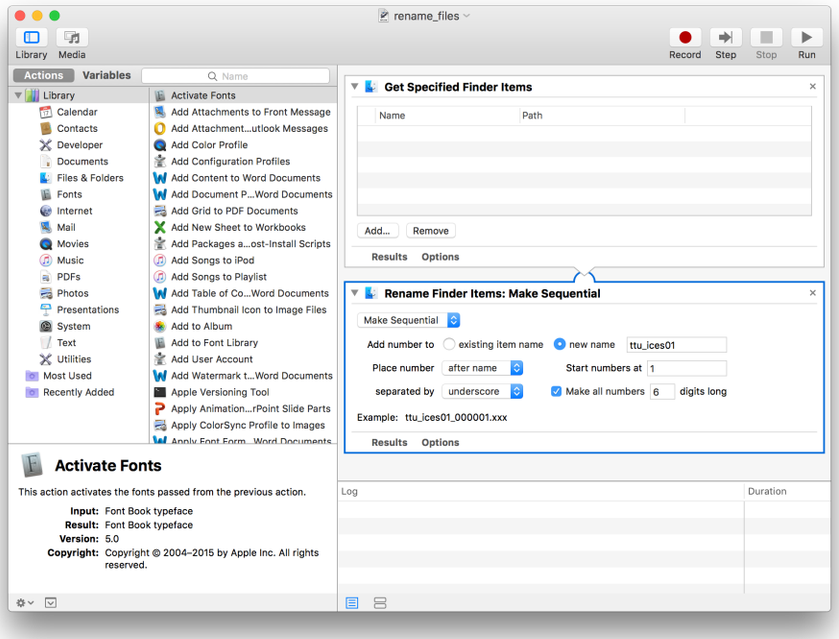
Necessity is a mother of invention, and it was a driving force for me. But there was something else that propelled me down my current path. A force even stronger than necessity.

Work Smarter Not Harder

I’m lazy … but in a good way. Not in the sense of being slothful or indolent.

Rather, lazy in the programming sense. In the programming paradigm, lazy refers to efficiency and avoiding repetitious tasks.

So the driving force for everything that follows came from a desire to work smarter, not harder.



This was an early attempt at automation. I needed to rename a lot of files based on the aforementioned naming standard. I did not want to rename files one at a time, so I used Automator to create a batch renaming task.

I also used Automator to create the folder structure for a new collection based on the archive structure presented earlier.

```
import os, shutil

baseDir = '/Users/christopher/Desktop/ttu_ices01/'
srcDirSuffix = '/metadata/display'
dstDirSuffix = '/content/display'
extensions = ['.pdf', '.csv', '.ppt', '.pptx', '.doc', '.docx', '.xls', '.xlsx']


lyst = os.listdir(baseDir)
for folder in lyst:
    print(folder + "\n")
    srcDir = baseDir + folder + srcDirSuffix
    dstDir = baseDir + folder + dstDirSuffix
    print(srcDir + "\n")
    print(dstDir + "\n")

    srcDirLyst = os.listdir(srcDir)
    for files in srcDirLyst:
        srcFile = srcDir + '/' + files
        dstFile = dstDir + '/' + files
        for ext in extensions:
            if files.endswith(ext):
                shutil.move(srcFile, dstFile)
```

But what to do with all of the files that I just renamed—move them into the correct folders by hand? This was clearly not the preferred solution.

So I turned to Matt McKinney who wrote a Python script. This moved all of my renamed files into the correct display folders based on file type.

At this point, I’m watching and learning. This was my first foray into Python.

```
import os, os.path, shutil

baseDir = '/Users/christopher/Desktop/ttu_wrc001_tiff/'
newDirPrefix = '/Users/christopher/Desktop/ttu_wrc001/'
newDirSuffix = '/content/archive/'

outputFile = open('/Users/christopher/Desktop/Archive/tiff_move.txt', 'a')

def moveTiff(dirName):
    lyst = os.listdir(newDirPrefix)
    tiffs = os.listdir(baseDir)
    for folder in lyst:
        dirLen = len(folder)
        newDir = newDirPrefix + folder + newDirSuffix

        for this in tiffs:
            tiffLenDiff = len(this) - dirLen

            if folder in this[:-tiffLenDiff]:
                tiff = baseDir + this
                newTiff = newDir + this
                shutil.copy2(tiff, newTiff)
                outputFile.write(tiff + " has been moved to " + newTiff + "\n")
                print (newDir)
            else:
                print ("no")

moveTiff(newDirPrefix)
```

Matt also created this script that moves all of the TIFF files to the correct archive folders.

I used Matt’s solutions, and all was right in the world.

Until last summer.

The Heidi Problem

If you don’t know Heidi Winkler, Heidi is our Digital Curation Librarian. Among other things, she curates the dark archive. One day, last summer, Heidi came to me and told me about her problems checking new collections to be placed in the dark archive.

She was having to go through collections to make sure folders and files were named correctly and in the right place. She was doing this manually.

She asked if I could help, and I said yes.

The Heidi Solution

Python

80 lines

procedural

command-line interface

checks for:

- missing folders/files
- incorrectly named folders/files

This is the Heidi solution.

It is a single Python script that checks for missing and incorrectly named folders and files and prints back the results.

Having solved the Heidi problem, it became clear that the problems she encountered were problems of human error. All of the errors being checked by this script were created by humans.

So, I wondered if there was a way to minimize human error earlier in the process?

It turns out there was.

The Lab Solution

Python

2 scripts

86 lines

procedural

command-line interface

creates archive folder structure for collection

renames files

This is the solution we implemented in the Digitization Lab at the beginning of the digital collection creation process.

With the archive collection structure in mind, we now create the collection folder structure before a collection is digitized. Once items are digitized, the files are renamed before they leave the Lab. This is all automated. There is no naming of files or folders by a human.

This solution eliminates all the errors that Heidi was checking for.

And Now For Something Completely Different

I currently do all of my programming in Python.

Some of you may not know that the Python programming language is named after the Monty Python comedy group.

This is my ode to Monty Python.

Although what follows is not completely different from the previous examples, it is, nonetheless, significantly different.

DSpace Batch Upload

What was different? DSpace 5 introduced batch uploading through the web UI. With this development came a request.

The Melanie Problem

Melanie Clark is our Architecture Image Librarian at the Architecture Library. She emailed me one day and asked if I could help her batch upload a collection to DSpace.

I said yes.

Now this wasn’t just a Melanie problem. This was a problem for everyone at my institution who works with DSpace. She was just the first to ask.

The Melanie Solution

I wasn’t sure what the solution would be, but there was one principal requirement. It had to be easy for Melanie and everyone else at our institution to use.

You see, I was envisioning a future in which I was the “batch upload guy,” and I didn’t always want to be the “batch upload guy.” I wanted everyone to eventually be their own “batch upload guy.”

Southwest Collections

I began searching for a solution. I went to Matt and asked for advice.

This is where Rob comes into the picture. This is where Rob used to work. This is the part of the presentation that Rob would be doing if he were here. Since he is not, I will briefly try to convey his work.

It turns out that, several years ago, Rob and Matt decided to begin doing batch uploads to the Southwest Collections DSpace. This was before the web UI batch upload option existed. They were uploading through the command-line. This was also before there was a tool to create the Simple Archive Format packages, or, at least, they were unaware of its existence.

```
<?xml version="1.0" encoding="UTF-8"?>
<dublin_core>
  <dcvalue element="contributor" qualifier="none">Norton, Charles G.</dcvalue>
  <dcvalue element="coverage" qualifier="spatial">Eastland County (Tex.)</dcvalue>
  <dcvalue element="coverage" qualifier="spatial">Ranger (Tex.)</dcvalue>
  <dcvalue element="date" qualifier="issued">????-??-??</dcvalue>
  <dcvalue element="format" qualifier="mimetype">application/pdf</dcvalue>
  <dcvalue element="language" qualifier="iso">en_US</dcvalue>
  <dcvalue element="format" qualifier="extent">Volume</dcvalue>
  <dcvalue element="publisher" qualifier="none">Times Publishing Company</dcvalue>
  <dcvalue element="subject" qualifier="lcsh">Ranger (Tex.) -- Newspapers.</dcvalue>
  <dcvalue element="subject" qualifier="lcsh">Eastland County (Tex.) -- Newspapers.</dcvalue>
  <dcvalue element="title" qualifier="none">Ranger Daily Times</dcvalue>
  <dcvalue element="type" qualifier="none">Text</dcvalue>
  <dcvalue element="type" qualifier="none">Newspaper</dcvalue>
</dublin_core>
```

I found out that Rob had developed a Python script that would create Simple Archive Format packages for their newspaper collections.

Because of the nature of these collections, the metadata was mostly identical for entire collections. Rob's script used a Dublin Core XML file with all of the metadata filled out except for the date that is highlighted here.

When the newspapers were digitized, the date was put in the file name. Rob's script took the date from the file name and populated the XML file.

This solution worked well for the newspaper collections, but it wouldn't work for us without a great deal of modification.

SAFBuilder

Rob told me about SAFBuilder.

SAFBuilder is a Java application that builds Simple Archive Format packages.

It works well, but it was not easy to install and use—or it wasn't as easy as I wanted it to be.

It did have ZIP functionality. However, it required command-line install and execution, and there was no GUI.

Also, files had to be in a single directory which wouldn't work for our collection structure.

SAFSomething

So I decided to create a new application that met my all of my requirements.

The requirements were, first, ease of use. I wanted a solution that would be easy for others at my institution to use. This means that it should be easy to install and use.

The application must be able to traverse subdirectories to find files. As discussed earlier, we had a collection structure with multiple folder levels. It would be very inefficient to have to copy the files into a single directory for the application to work.

When uploading through the DSpace web UI, you must upload your SAF packages as ZIP files. You must also limit the size of a single ZIP file to 2GB.

Since we would be uploading through the web UI primarily, this application needed to have those features.

I was excited to start on this project. Not only would this project produce a usable product for our institution, but I might learn a little something in the process.

DSpace Simple Archive Packaging Utility

One day as I was working away, Joy Perrin sent me an email with a link and the subject line:

“FYI - someone wrote about batch uploading in DSpace 5”

I clicked the link, and it turns out that someone had indeed written about batch uploading. They had also written a Python version of SAFBuilder.

It turns out, however, that this solution did not meet my requirements.

- Command-line execution
- No GUI
- No ZIP functionality - unlike SAFBuilder

For our needs, this wouldn’t work. So I continued my work.

Then the the program for TCDL came out.

SAFCreator

So what did I see? A workshop by James Creel about his version of SAFBuilder.

I actually went to James' workshop yesterday, and his application is very good.

It has a GUI and it will traverse directories which met two additional requirements.

But it still did not meet all of my requirements.

- No ZIP functionality
- Command line install and execution

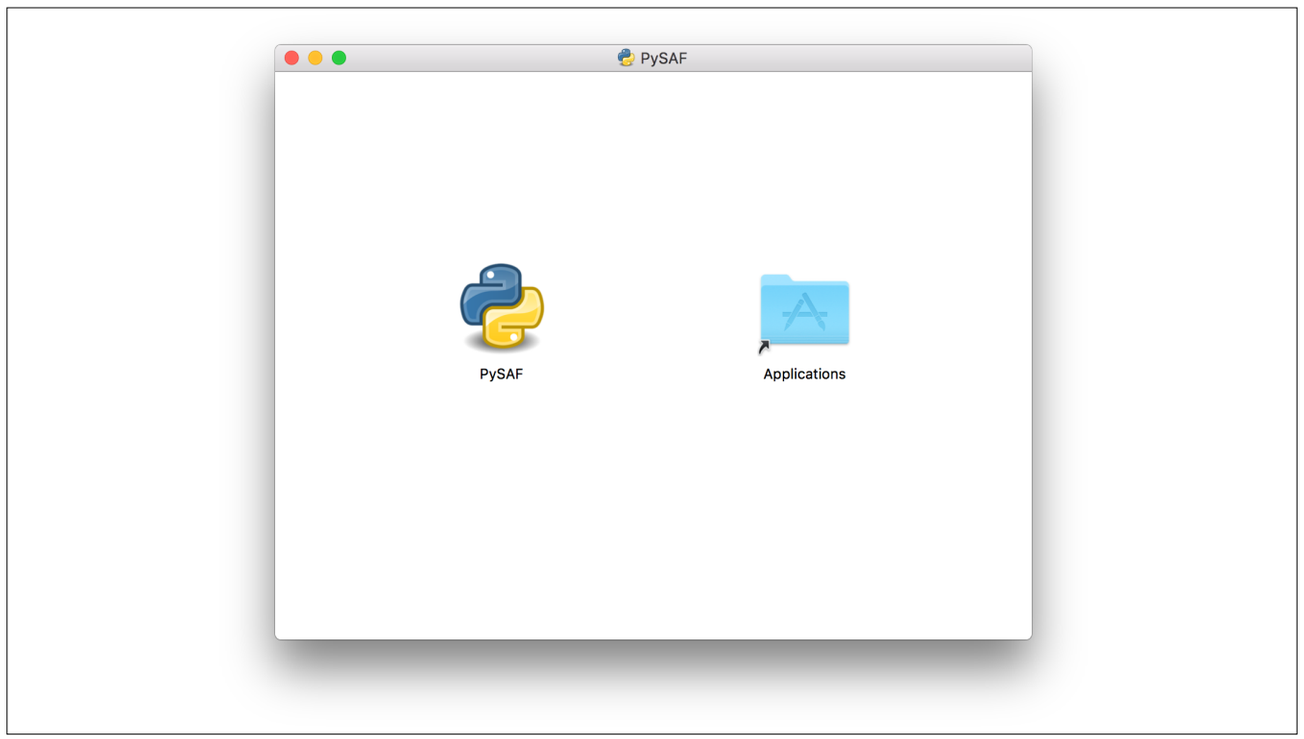
Besides, at this point, I had come so far, and I wasn't going to stop. And this was a learning exercise.

PySAF

Introducing PySAF. PySAF is an application for creating Simple Archive Format packages for batch upload into DSpace. It is written in Python, hence the Py in PySAF.

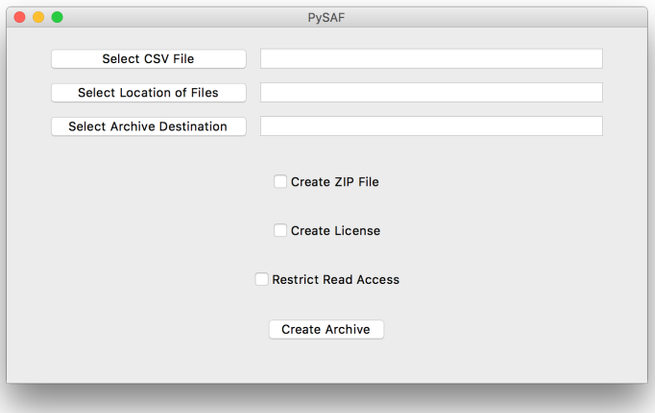
I was thinking of naming it something along the lines of SAFBuilder and SAFCreator, but I didn't care for the remaining synonyms.

On a side note, the screen shots in the rest of this presentation are from the OS X version, but be assured, there is a Windows version as well that looks very similar and has the same functionality.

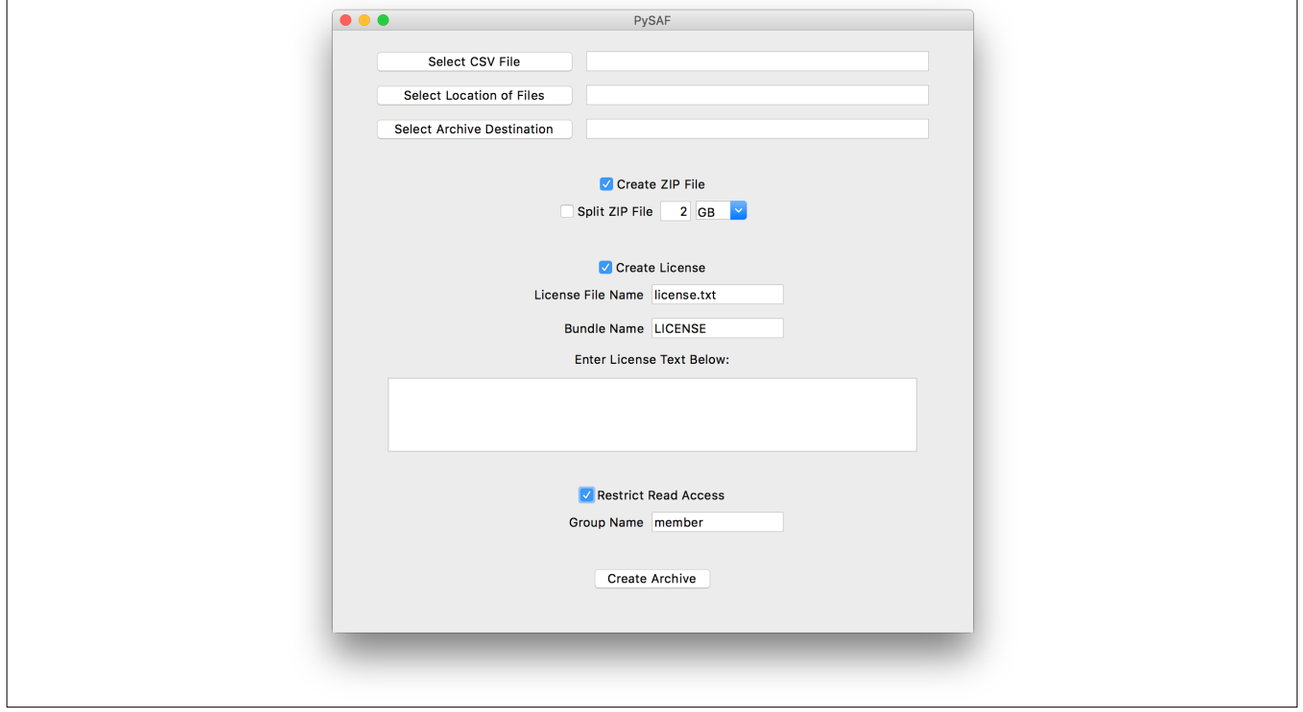


PySAF is easy to install.

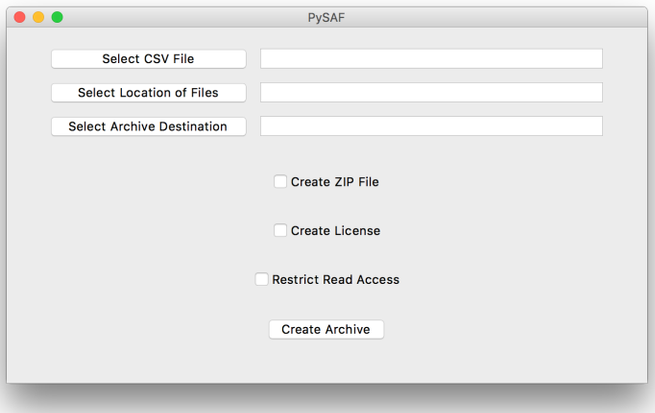
For Windows, the installer runs as any other Windows application installer. It will populate your Programs list in the Start Menu and will even give you the option of creating a Desktop Shortcut.



PySAF is easy to use.



PySAF has all of the desired functionality. At least, it has all of the original desired functionality.

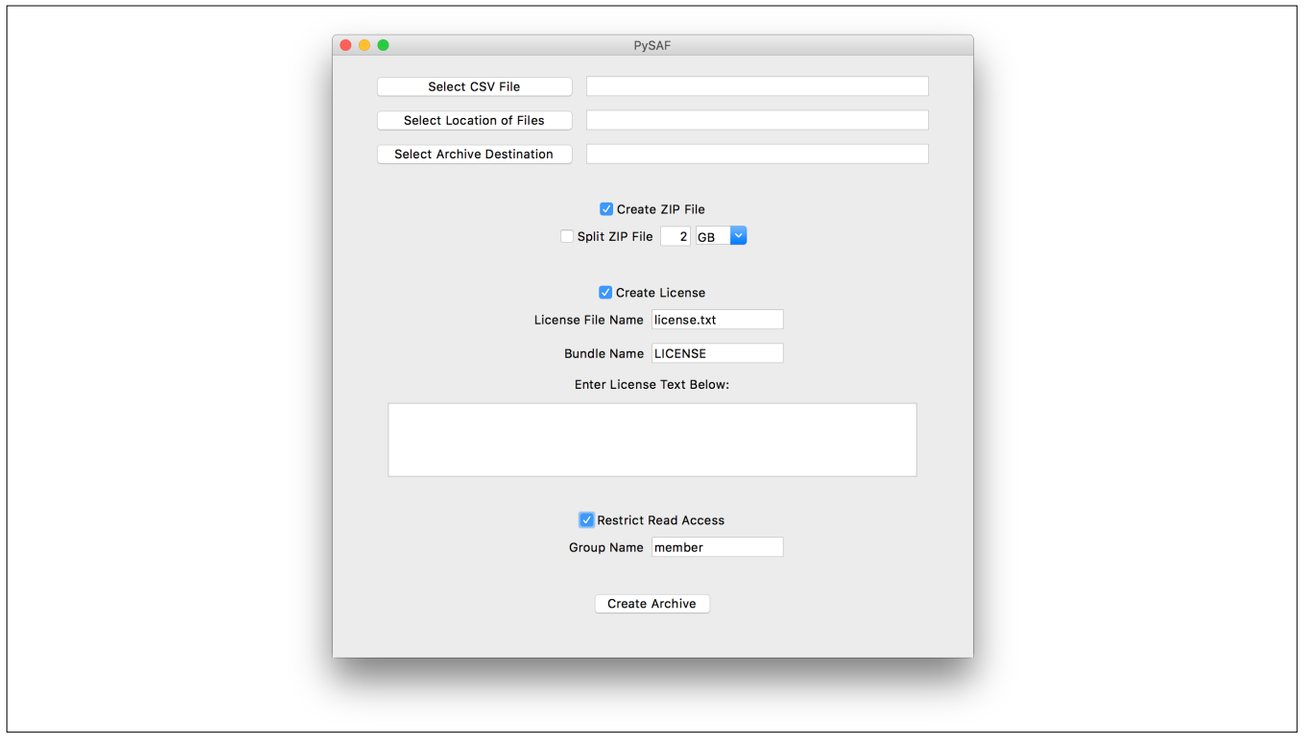


So this is the first screen you see when you launch the application.

Select CSV File
: This file must be encoded as UTF-8.

Select Location of Files
: Files don’t have to be in the same folder, just the same base directory.

Select Archive Destination
: This is the output location for your archive package.



Extended functionality.

Create ZIP File

If you select Create ZIP File, a ZIP archive will be created of the SAF archive. You can choose a maximum ZIP file size limit by selecting Split ZIP File. You can set this to any unit of MB or GB.

Create License

If you select Create License, a global license file will be added to each item in the SAF archive.

Restrict Read Access

If you select Restrict Read Access, you can restrict bitstream (file) access to any group defined in your DSpace.

With A Little Help From My Friends

Lest you think I produced this in a vacuum, I must acknowledge the contributions of my colleagues.

James Creel - I just met James yesterday, and although he wasn't aware of it, he did influence the functionality of this application through his application, SAFCreator.

Rob Luttrell helped me think through the design and particularly the functionality. He also helped greatly with testing. Rob is quite talented in the art of breaking things. But as frustrating as that can be at times for a developer, he helped make this a better application.

Matt McKinney, well, this application certainly wouldn't work as well and might not work at all, if not for Matt's contributions. Matt contributed significantly to the code, code design, and functionality.

I’m Not Happy

Or rather, I’m satisfied.

Known Issues:

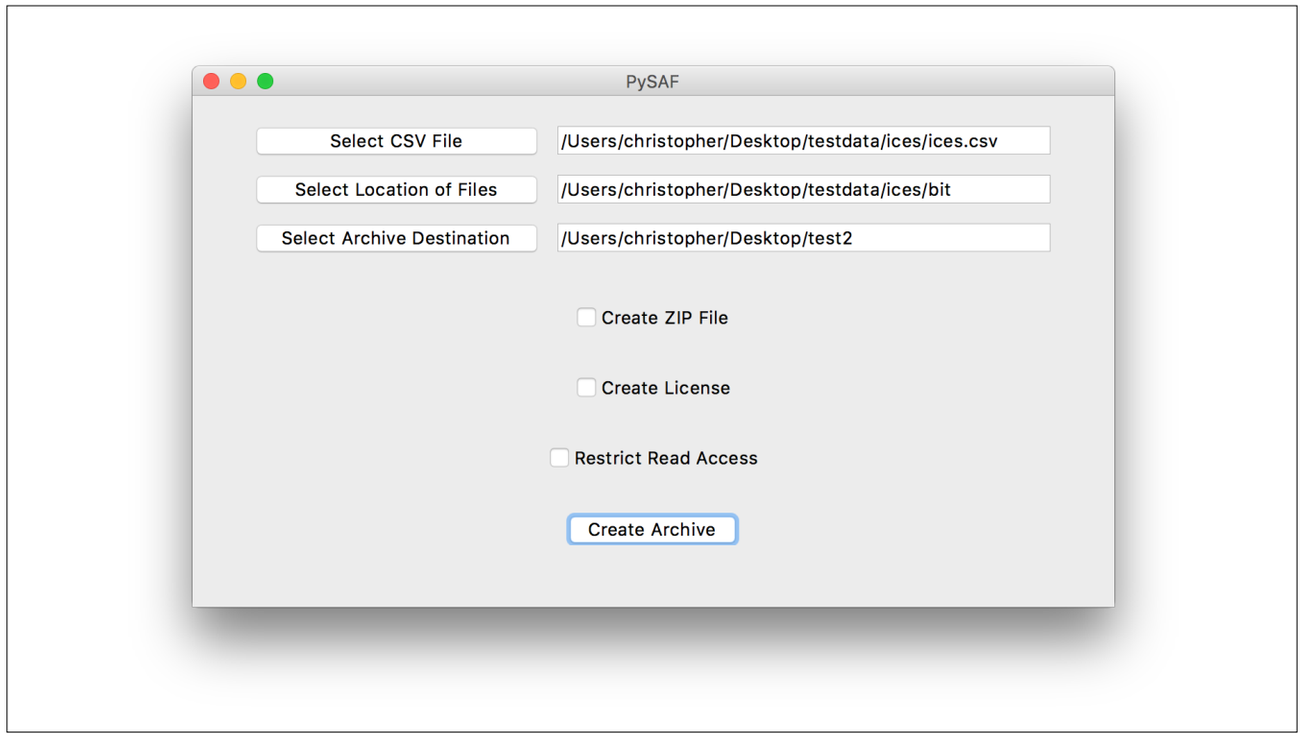
UTF-8 - As of now, your CSV file must be encoded as UTF-8. Character encoding is fraught with problems. This did not become a problem until I started testing on Windows. Windows uses a default character encoding other than unicode. So, to fix some unicode errors that were creeping up, the application requires UTF-8 encoding. If your file is not encoded correctly, you will receive a Unicode error message. I am working on a solution for this, but it is not currently working satisfactorily, so it is not included in this release.

GUI Freezes - This does not affect the functionality. It just means that you cannot use the GUI while the process is running. For some reason I have yet to discover, the GUI module I used is not allowing me to thread my processes the way I would like to. This is either a deficit in the module or my abilities as a programmer. We’ll assume it’s the former.

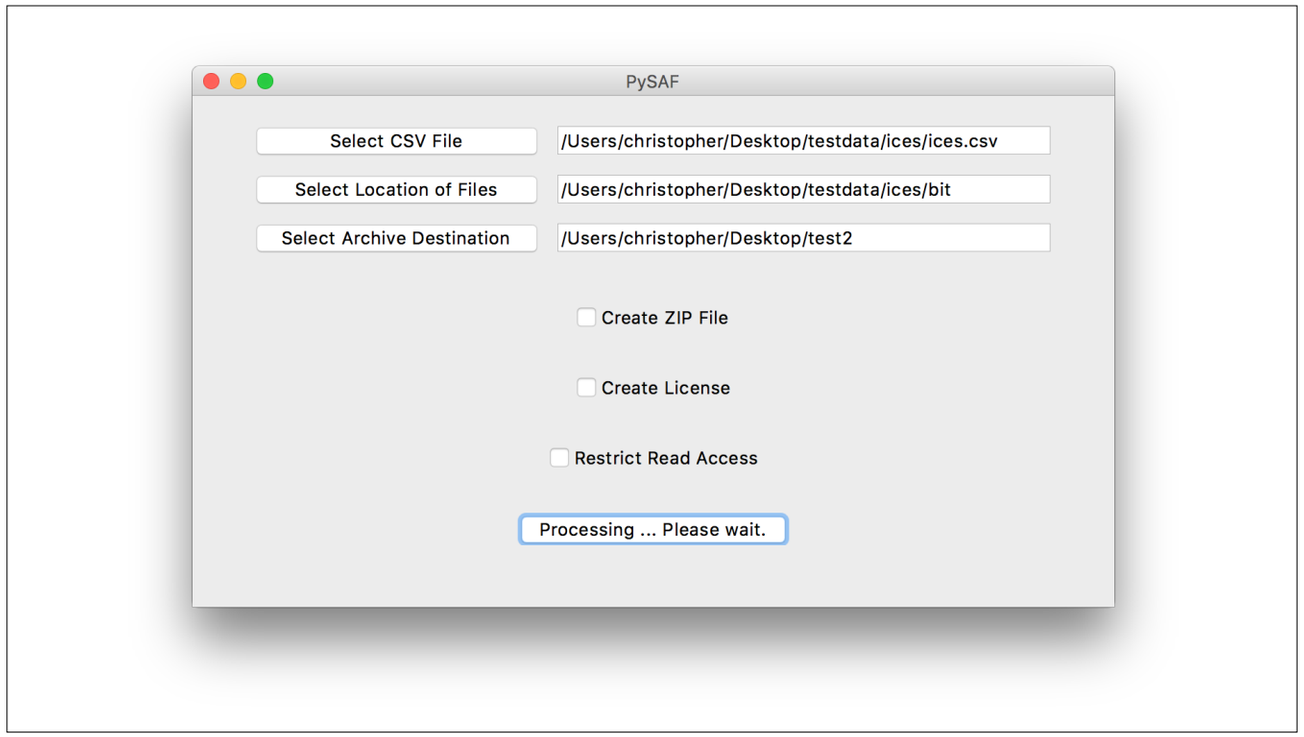
Progress Bar - There is no progress bar. I really wanted a progress bar. This is related to the GUI freeze issue. However, there is a workaround. The “Create Archive” button will change to “Processing…Please wait.” to alert you that the process is still running. When it is finished, it will change back to “Create Archive.”

OS X Font - There is a problem with the font in the OS X binary. It is a little fuzzy.

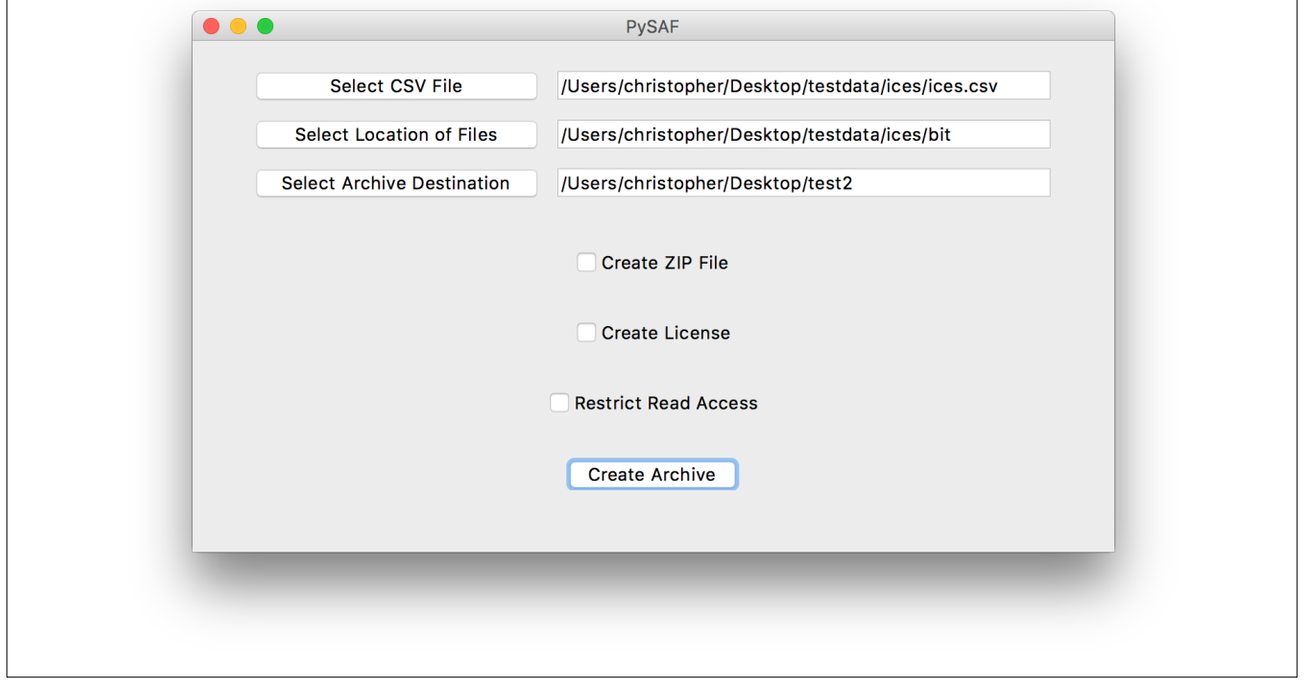
Icon - I don’t like the logo. It is currently the Python logo. I may try to come up with something unique.



I want to show you what happens with the Create Archive button. This is what it looks like before you click the button.



This is what it looks like while the application is processing your files.



And then it changes back to Create Archive when it is finished.

Again, it’s not the preferred solution, but it is a solution.

Other Noteworthy Features

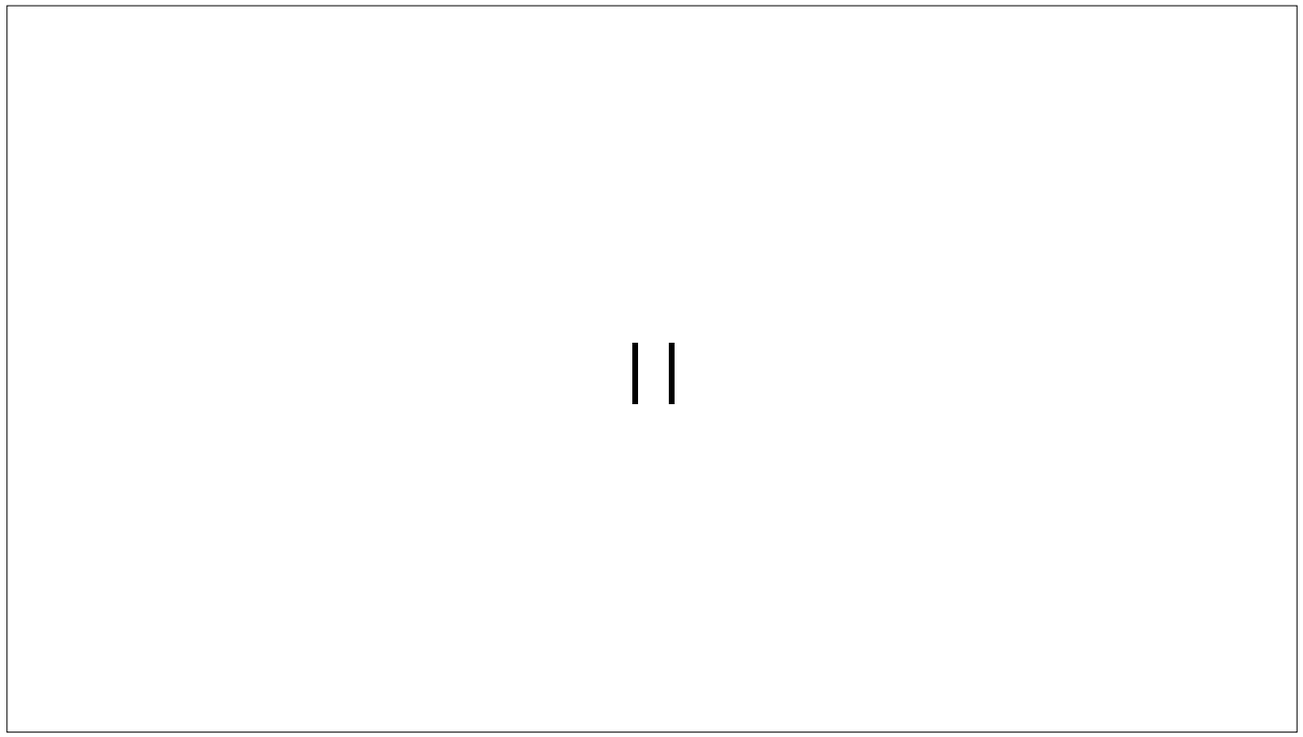
There are a few other features I would like to point out.

filename	dc.title	dc.contributor.creator
ttu_sasser_000001.jpg	Bahai Temple	Elizabeth Sasser
ttu_sasser_000002.jpg|ttu_sasser_000003.jpg	Hyatt Regency Atlanta	Elizabeth Sasser
|ttu_sasser_000004.jpg	Hyatt Regency Atlanta	Elizabeth Sasser
ttu_sasser_000005.jpg	St. Luke's Methodist	Elizabeth Sasser
ttu_sasser_000006.jpg|ttu_sasser_000007.jpg	1st Christian Church	Elizabeth Sasser
ttu_sasser_000008.jpg	1st Christian Church	Elizabeth Sasser
ttu_sasser_000009.jpg	St. Luke's Methodist	Elizabeth Sasser
ttu_sasser_000010.jpg	St. Luke's Methodist	Elizabeth Sasser

Your CSV file must have a specific structure. The top row must have column names.

The first column must be named “filename” in all lowercase. The other columns should have the Dublin Core element name.

If you look at the filename column, you will see that some rows have multiple file names, and although it is not easy to see in this screen shot, there are a couple of characters between the file names.



This is the double pipe. If you have looked at a DSpace metadata export, you may have seen this. This is how DSpace exports multiple entries for the same metadata field. But the double pipe is not just for DSpace, you can use it too.

```
file1.pdf||file2.pdf
```

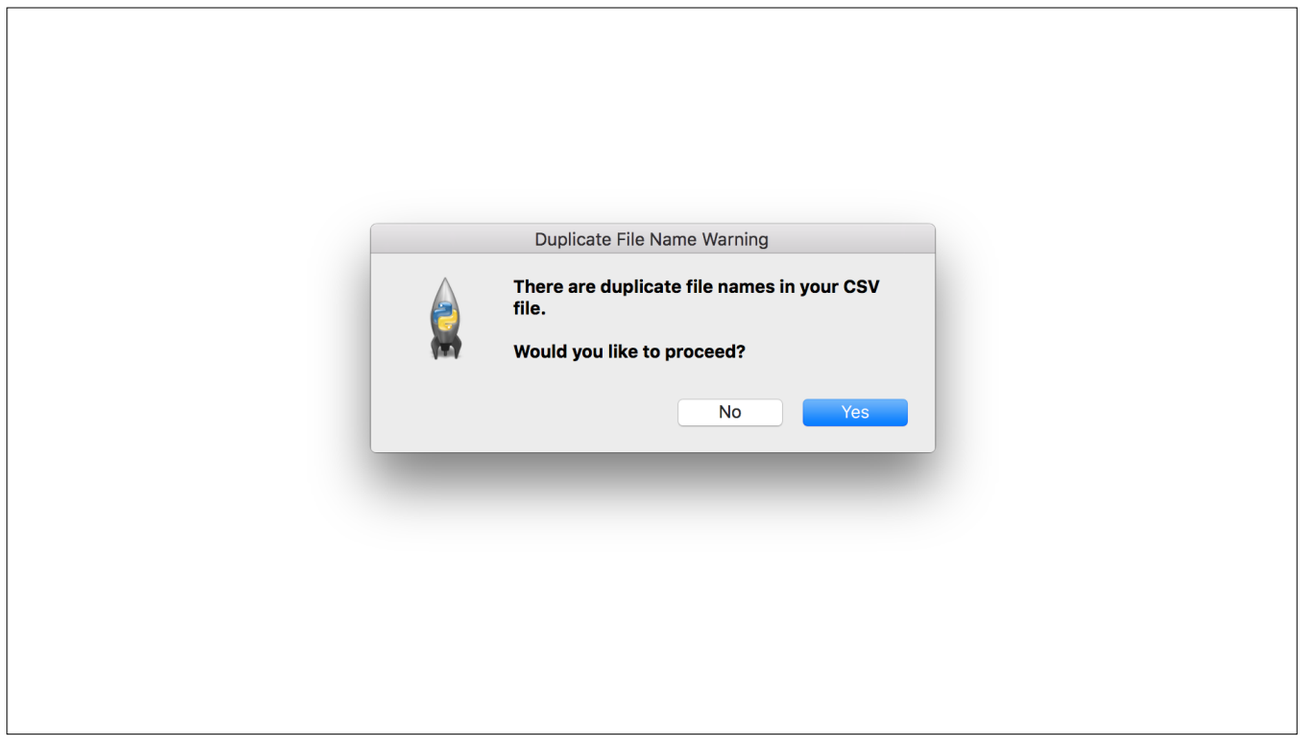
If you have multiple files for a single record, just put a double pipe between them.

```
author1||author2
```

You can also do this with any other field in your CSV file. PySAF will create the correct XML metadata file.

Errors

There are many error messages that you could encounter while using this application. They should be self-explanatory, but there are a few I would like to discuss.



This functionality was a feature request from Joy Perrin.

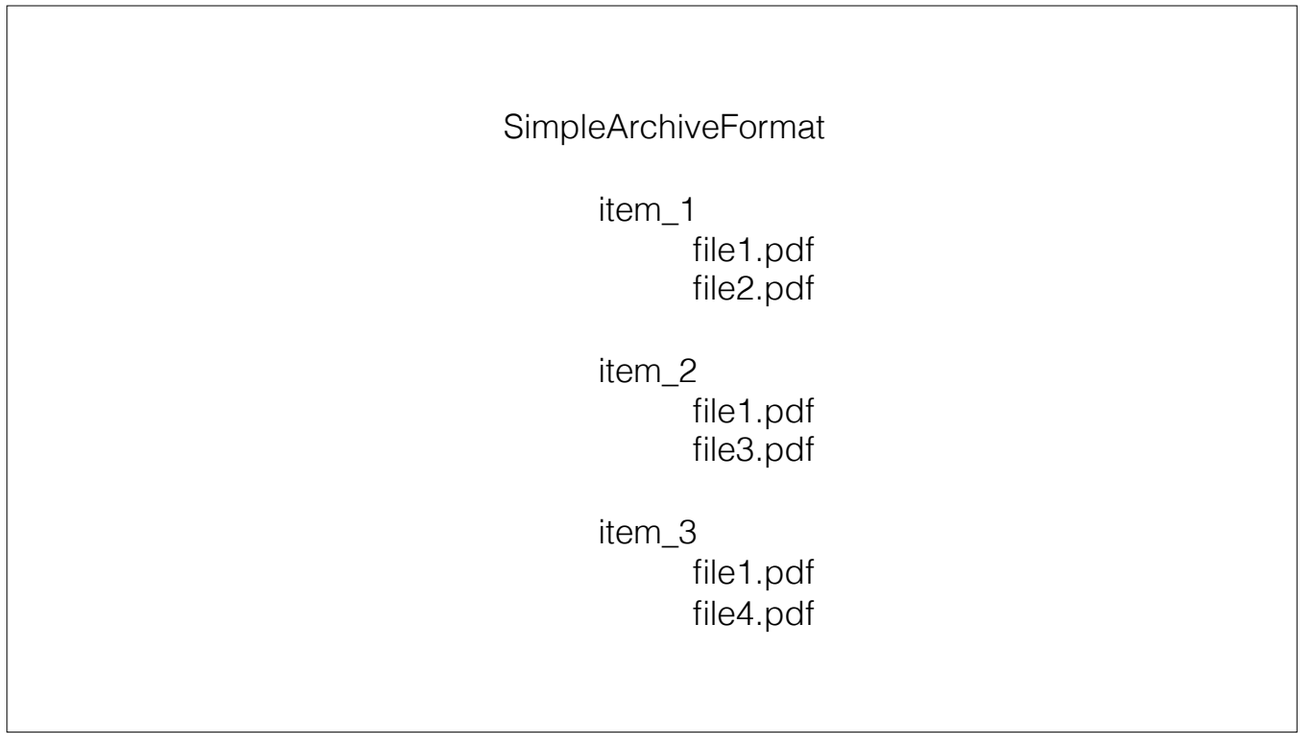
PySAF will allow you to have duplicate file names in your CSV file. However, to make sure this is your intent, you will receive a warning.

If you select Yes, the file or bitstream that corresponds to that file name in your CSV file will be copied for each item record in which that name appears. It will make more sense if I show you.

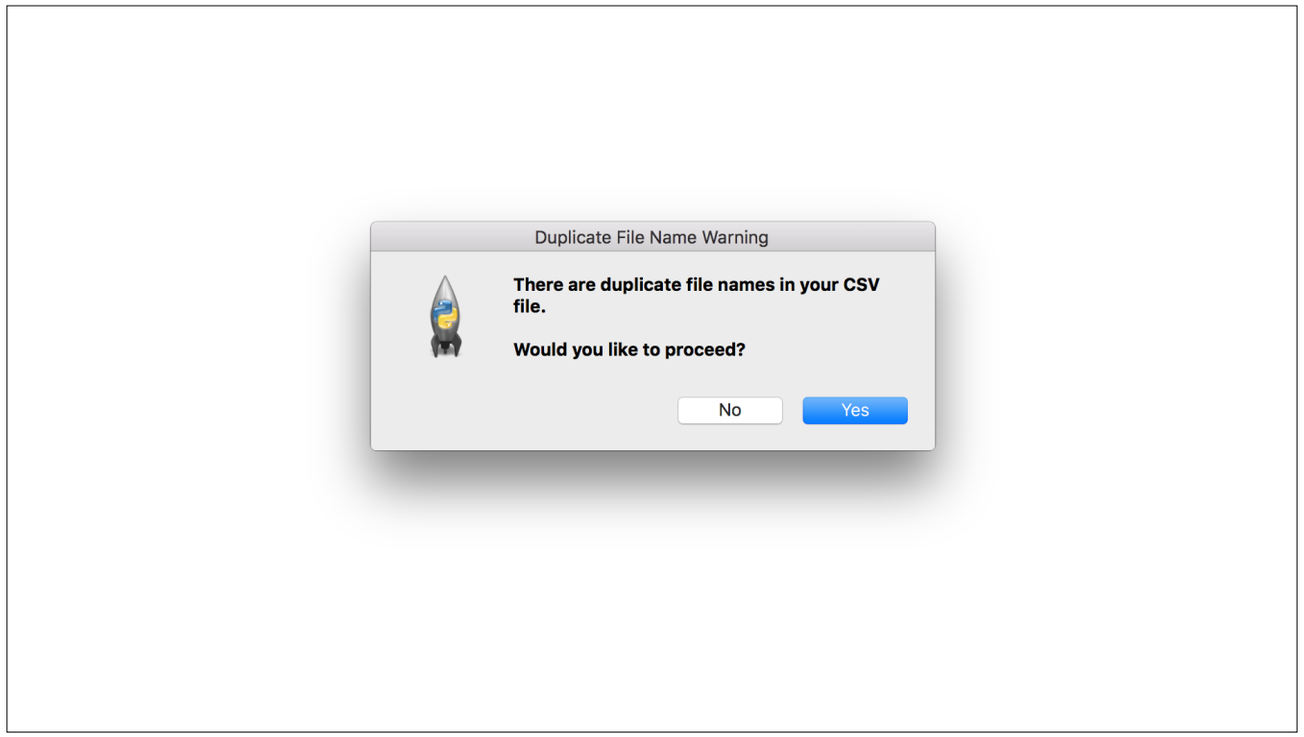
filename	dc.author	dc.title
file1.pdf||file2.pdf	Arthur One	La Ti Da
file1.pdf||file3.pdf	Arthur Two	Ti Da La
file1.pdf||file4.pdf	Arthur Three	Da La Ti

Say you have a file (file1.pdf) that you want to include in multiple records. As you saw earlier, you can use the double pipe to enter multiple values in a single field.

But what will happen if you do this?



So this is the result. Of course, all of the other files will be in there, too.

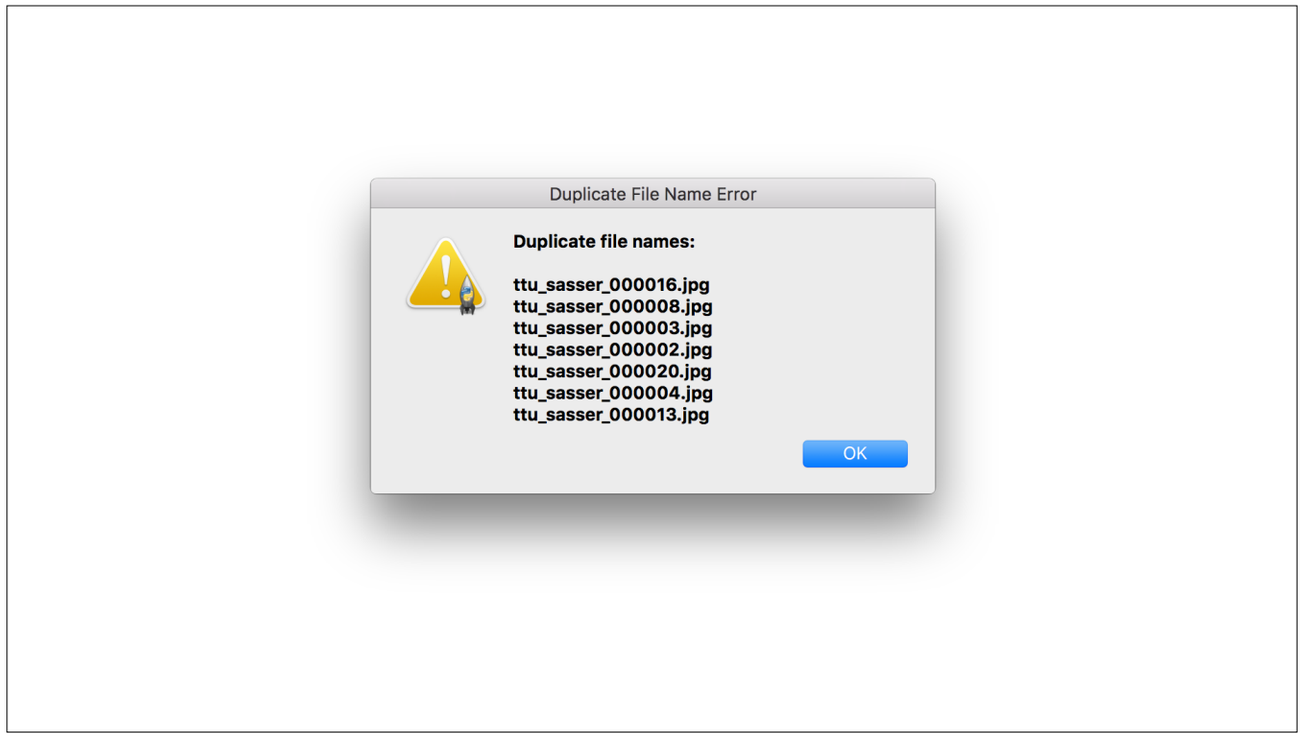


So that’s what happens if you select Yes.

What if you select No?

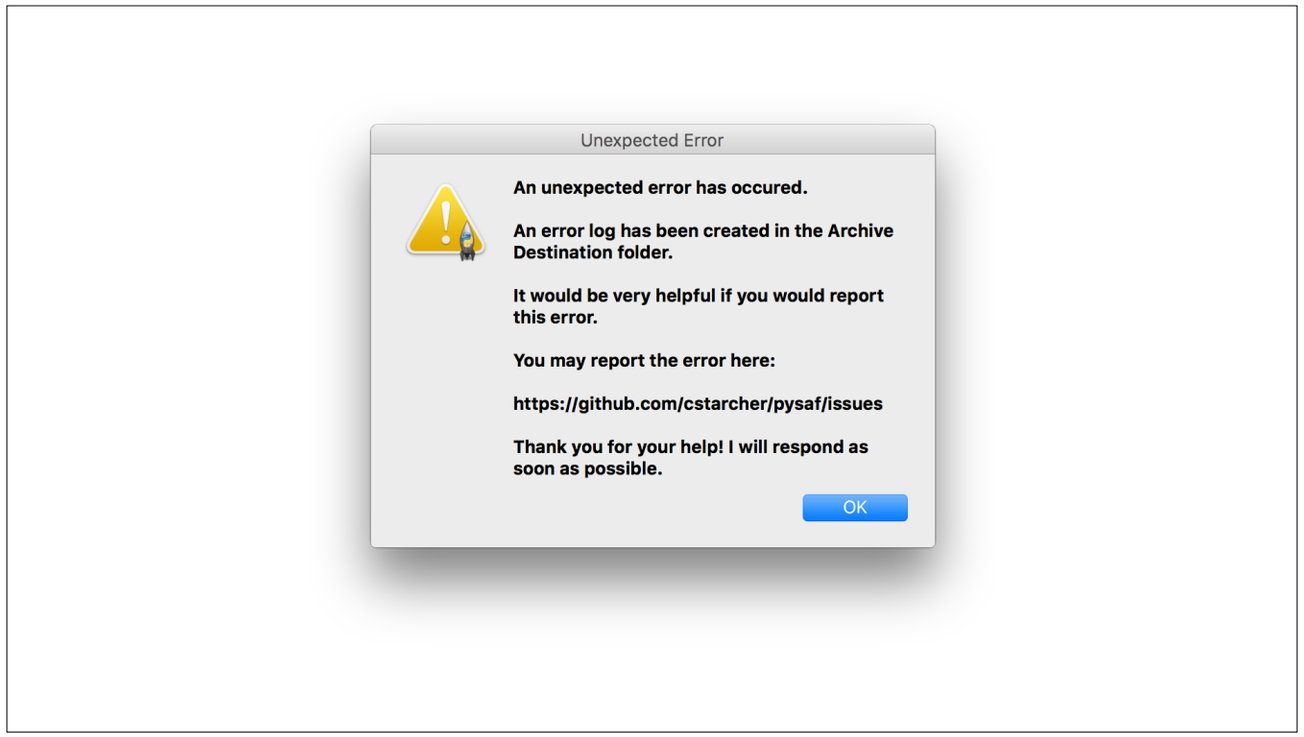
If you did not intend to have duplicate file names in your CSV file—you just got a little copy/paste happy—then you will want to select No.

If you select No …



You will get this error message reporting the duplicate file names. This allows you to go back and fix this in your CSV file. It even tells you which files names were duplicated so you don’t have to guess.

A similar message will appear if you have multiple files with the same name, or if there are any files missing that are listed in the CSV file.



There is one other error message that I want to point out. This is the most important to me. All of the other errors that you would encounter mean that you have done something wrong. This one means that I have done something wrong.

If you see this message, that means an error has occurred that I have not encountered and the application does not know how to handle. This means even Rob didn't find it.

This exception creates a log file that won't mean anything to you, but will mean a lot to me. I would greatly appreciate it if you would follow the instructions and report the error. I did not find this error in my testing, and if you don't report it, I won't know about it, and I can't fix it.

https://github.com/cstarcher/pysaf

Here you are. My gift to you.

There is a Windows 32-bit binary (.exe), a Windows 64-bit binary, and an OS X (.dmg) binary available for installation.

If you are running this on Linux, at this point, you must download and run from source.

You may run from source on any platform. If you do, I highly suggest that you install Python 3.5 or higher. It is possible that it may not work correctly on earlier versions, but I know for sure it will run slower on any version earlier than 3.5.

All of the installation and use instructions can be found in the README on GitHub.

Feature Requests

Within reason and as time permits, I will entertain feature requests.

Alternately, if you know Python, feel free to contribute a branch on GitHub.

I would also be glad to have comments about the design. If the wording of an error is confusing, let me know. I don't work for Microsoft, so I have no mandate to write confusing error messages.

I really am happy to have your comments and feature requests. And you never know, they could end up in …

PySAF 2

Already?

Well, not yet. However, as I pointed out earlier, there are some thing I am not happy with, and I want to grow my programming skills.

So, yes, I anticipate that this will be coming in the future, but I have no timeframe.

Why Python

If you decide to pick up the mantle of automation and take a similar journey, there are many programming languages to choose from.

I chose Python. There are lots of reasons to use Python, but I want to point out a couple in particular.

0 to OO

You do not have to go from zero to object-oriented programming to produce useful code.

Community

The Python community is large and there are many resources available.

General Resources

Stack Overflow

Youtube

Books (e.g. Safari, etc.)

Tutorials (e.g. Codecademy, Lynda, etc.)

Source Code

These are some of my most used resources that would be beneficial for whichever language you choose.

Reading source code from other projects is one of the best ways to learn how to code.

Python Resources

Google/MIT

Raymond Hettinger/Ned Batchelder/David Beazley

Idiomatic Python

PEP 8

Source Code

These are great resources if you choose to use Python. Both Google and MIT have excellent online courses for beginning programming with Python.

Any video or written information you can find from the names listed here will be useful.

Learn to write idiomatic Python.

Spend time understanding PEP 8.

Again, reading source code is an excellent way to learn.

Encouragement

Write code
Refactor
Make mistakes

Although your aptitude for programming will determine how far you go, success can be defined on varying levels.

christopher.starcher@ttu.edu

https://github.com/cstarcher

Again, if you run into any problems or have feature requests, let me know.