BTD Importer

CS 4624

Virginia Tech, Blacksburg

May 8, 2014

Clients: Paul Mather, Zhiwu Xie, Jie Chen

By: Nathanael Bice, Scott Brink, Adam Piorkowski

Table of Contents

Executive Summary

The BTD Importer is used to importer Bound Thesis Dissertations to the Electronic Thesis Dissertation Database. The process involves taking a hard copy thesis and scanning it into PDF form. Once in PDF form, the importer script would locate the new PDF and extract its library call number, which is located in the PDF file's name. Using the call number, the importer script would fetch the metadata of the thesis, such as title and author, by scraping the metadata using AirPAC Classic. The PDF would then be uploaded to the ETD database along with its metadata.

The BTD Importer deliverables listed a new importer script that would take new PDFs and look up their metadata using the Sierra APIs to access Addison directly, then taking that metadata and constructing an XML file containing the data. The script would then move the PDF and the new XML file to a new output file structure, which would later be read, by another section of new the importer process. That final section would then upload the PDF and XML file to VTechWorks. The project would require PHP skills which Nathanael had and SQL skills which Adam and Scott had knowledge of, so our group felt like we could complete the project satisfactorily. The project spec also listed the project as being high impact as our work would be used to import roughly 13,000 BTDs into VTechWorks.

We completed the project by splitting up the work amongst the group and meeting weekly to discuss milestones and our next goals. We decided to stick with using PHP, as that was what the original importer script was written in. The PHP libraries made it very straightforward to construct an XML file and a directory structure.

Background

Our group selected the BTD Importer project based on the required skills list, and the fact that we knew the project would be used extensively upon completion. Our group split the work up by components of the new script. Adam wrote rigorous exception handling, Nathanael wrote the code to create the XML file, and Scott wrote the portion that moved the PDF and XML file to the correct directory in the output. We met weekly to discuss progress and if we needed help with anything going on. At first we started recoding everything from PHP to Java because we felt we all knew the Java libraries better. Quickly we realized that it would take one hundred lines of code to accomplish what could be accomplished in 2 lines in PHP. With that we decided to stick with using PHP and we brushed up on the PHP libraries. Meeting weekly encouraged us to not get far behind from our timeline and it ensured we did not push large parts of the project off.

We decided to split the importer script into 4 main portions. The steps included fetching PDFs and their call number, fetching metadata, building the XML file, and creating the output directory structure. We went in order working on each step one-by-one, and then worked to create robust error handling to ensure there would be no problems when running this script daily. Once we got further into the semester, we began weekly meetings with Jie Chen who will maintain the script after the semester ends. He provided feedback on our current work, gave us suggestions for improvement, and also assisted in creating test cases. Overall the project was challenging and rewarding, knowing that the outcome of our work will be used frequently.

User's Manual

To use the importer script, you will need to know the source directory to be scanned for new PDFs. Once you have that information, you will be able to run the script to find new PDFs in the source directory and fetch their metadata. Run the script using the command 'btd_importer.php -q production -d {source_folder}'. Running this will then scan for all PDFs and attempt to fetch their metadata. On success, the PDF and the new XML file will be moved to an output directory. There will also be a log file created to list all successes and failures. PDF's that failed during the process are moved to a failed folder to be manually looked at and imported. The output directory can then be used as input to the next portion of the new import process to be uploaded to VTechWorks.

Developer's Guide

Functional Description

The BTDImporter application consists of 2 major parts, the scanner and the importer. The scanner is a shell script that is used to feed the relevant directories to the importer script. The importer script takes a directory or file and prepares each PDF file with the appropriate naming convention for importation into VTechWorks. The scanner is contained in the file btd_scanner.sh and the importer is found in the btd_importer.php file and the lib_addison.php file.

Scanner

The scanner is a fairly simple program. It first checks to see if there is an existing process that is running the scanner. If so, the application exits. Assuming there is no existing process running the application, a find command is fed into a grep command to find all directories that follow the appropriate naming convention and have been touched in the past day. It then iterates through each of these directories and calls the importer script on them.

Importer

The importer application is where the bulk of the work for the BTDImporter occurs. The application starts in the btd_importer.php file. The first part of the application checks to ensure that the functions that are required for this code to run are available. This is accomplished in the "check_required_functions" function. The next sequence consists of parsing the command line arguments. If any of the arguments are invalid, it echoes the usage instructions and exits. The command line arguments are fairly simple. First, either the q or s flag must be used. The q flag signifies that the logging information will be echoed to the console. The s flag means that only major error information will be echoed to the console. After this, the mode of operation is specified, "test" or "production". If the application is in test mode, there will be no changes to any data and no files will be created or moved at any point, other than the log file. In the production mode, the application will function as expected, creating the output directory and moving and creating files as necessary. More on that will be described later. The next argument is either the f or d flag. The f flag should be followed by a specific file for processing whereas the d flag will be followed by a directory. If a directory is specified, then the files contained in the directory will be read into a list. Finally, the last arguments specified are key-value overrides for entries in the metadata records. These are stored to be applied later. After this, the files to be processed are iterated through and processed in sequence.

The processing of import files is broken up into four logical modules. This first module is the name parser. This module compares the name of the current file against a regular expression. This regular expression makes a number of checks against the format of the name of the file. The check will fail if the file is in an incorrect format or if the file does not have a PDF file extension. If either of these conditions are not met then the file will not be considered for importing and will be ignored. If the file passes the check, then the last four digits of the filename, excluding the file extension, will be extracted for later use. This section is contained entirely in the global level of the btd_importer.php file. This section also checks to make sure that the file being processed is not currently being written to and that the file has not been processed before. The duplicate check is done through checking to see if there exists a file with the same name but the extension "import_success" in the log directory. If such a file exists, the file is ignored and an appropriate message is output.

The second module is the metadata fetching section. This module takes the last four digits of the file name, which is the library call number, and fetches the bib number from the AirPAC classic service. From this, the bib number is used to query and pull the complete library metadata record from the AirPAC service. This metadata record is then converted into an array of data. After this, the command line overrides previously mentioned are applied to the array. The last portion of this section is to validate that all required fields are included in the record. Currently, the only required field is "title" but the structure is in place to handle any number of required fields. This all occurs in the first portion of the import_btd function.

The third module consists of the building of the output XML file. Initially, a root dublin_core element is created. After this, a child node is added for each of the potential values in the schema if the values of that node are stored in the record received from the AirPAC service. In addition, some of these child nodes are static and are added to every XML file. This code is found in the middle of the import_btd function.

The fourth and final module is the section that moves the file being processed to the output directory and writes the corresponding output files. The first step is checking to see if the application is in production mode. If it is not in production mode, then this module is skipped. The next step is writing the "import_success" file. This file has the same base name as the file being processed but has the file extension ".import_success". This is the file that is used in the first module to check for duplicate import files. This file contains all of the information that is stored in the debug buffers, including debug information and the record that was pulled from the AirPAC service. This file is written to the log directory, which is nested in the input directory. If this directory does not exist, it is first created. After this, the program checks to see if a pending directory exists inside of the input directory. If it does not, then the directory is created. A directory within the pending directory is created for each file that is processed. The directories are of the format "item_#" where # is the next highest number that has not been used, starting at 0. The file being processed is next moved to its folder. After this, the contents file is opened and written to the subdirectory. In it, there is a single line for each file to be stored in the subdirectory, the processed file and the Dublin core XML file. Finally, the XML file that was constructed in the third module is written to this directory. This module takes up the last portion of the import_btd function.

Throughout each of the above modules and the initialization code, a log file keeps track of all debug information and writes it to a file. This log file has the naming convention of "YYYYMMDD.log", using number format. In addition, if any errors are encountered throughout the program while in production mode, the file being processed is moved to a folder, "Failed", for manual processing. In addition, any corresponding output files that may have been created, such as "Contents", are deleted. The rest of the program merely tracks the success and failure of each file. This information is used in a structure that is set up to send out notification emails, but those email do not currently send, due to a lack of direction about them.

File Description

Source Files

btd_scanner.php

This file contains the shell script used to run the automated scanning and importing process. This script first checks to make sure that another instance of this script is not already running. After that, it gets a list of all the directories that have been touched in past 24 hours. The directories also must follow a specific name convention. The script then passes these directories to the btd_importer.php script, one at a time.

btd_importer.php

This file contains most of the logic used to import each bound thesis. In summary, this script checks if each file in the directory is valid and then moves it to the output directory while creating an XML file with the PDF file's metadata. The in-depth description of this process is described in the "Importer" section of the functional description above.

lib_addison.php

This file contains helper functions that are used in the btd_importer.php and lc_lookup.php scripts. These functions are used to retrieve and manipulate data relating to the Addison library. The functions are retrieving the bib number based on a given call number, retrieving the library xrecord based on the bib number, converting the xrecord into an array, and querying against the Addison system, though there are several others.

lc_lookup.php

This is used to test the connection against the Addison system. It takes a single command line argument of the call number to look up. After this, the array of the library xrecord is printed to the console.

Output Files

dublin_core.XML

This file is used to store the metadata for the PDF file that is stored in the same directory.

contents

This file is used to describe what files reside in the current directory that should be packaged together when brought into the VTechWorks system.

Lessons Learned

Timeline/Schedule

At the beginning of the semester we developed a tentative schedule and timeline of tasks to complete by the end of the semester to accomplish our client's goals and our own goals for the project. Our initial timeline was based on the input and information we learned from meeting with our client, Paul Mather, initially on February 5th. We tried to follow this schedule the best we could throughout the semester, but as we ran into problems or goals of the project shifted our schedule adjusted to deal with these changes. Our final timeline after completing the project and meeting our client's goals as well as our own can be seen below.

- Met with client to discuss project and milestones - Feb. 5
- Discussed and assigned project roles - Feb. 18
- Reviewed provided code to determine what changes are to be made - Feb. 18
- Began work on metadata fetching system - Feb. 28
- Met with client to discuss progress - March 6
- Created working version of importer script - March 6
- Finished Stage 1: Thesis Call Number Parsing - March 15
- Created file structure output - March 21st
- Started XML File creation - March 29
- Finished Stage 2: Metadata Retrieval - April 5
- Finalized XML file creation - April 15
- Finished Stage 3: XML File Parsing & Creation - April 17
- Began error handling - April 19
- Finished Stage 4: Moving XML files and PDF to output directory - April 20
- Began writing test cases - April 21
- Met with Jie Chen (Who is maintaining project after completion) - April 21
- Added suggestions from Jie to the script - April 22
- Finalized Error Handling - April 25
- Finished Project Presentation - April 26
- Finish testing - April 27
- Met with Jie Chen for final project approval - April 30
- Received final approval - April 30
- Final Presentation - May 1

As I previously stated we initially met with Paul to discuss what his expectations for the project as well as how he wanted us to go about implementing the new BTD importer script based off of the old script. After we had an advanced understanding of the project and what the end product should look like we met as a team to come up with a plan of how to tackle our script and came up with a design. We split the work among our team into equal parts for each of us to work on. During our initial meeting to assign work roles and design our project we used the previous BTD script as a learning tool to see how the process worked in the past. We learned the ins and outs of the old script to see what pieces may be useful and other pieces that we could improve upon in our importer. This process took some time, as we did not have proficient knowledge of PHP.

After having an understanding of the old importer, the first task we set was creating a working script that would fetch the metadata of the BTDs from the Addison library system. During this stage of the project, we were still using the old system, AirPAC Classic, to scrape the metadata off of the Addison page instead of the Sierra API because we still didn't have access to that system. One end goal of the project was to replace the scraping system with the Sierra API. We met with Paul Mather again after we had worked on the front end of our project for fetching metadata to discuss our progress as well as to clarify some of the project requirements and to inquire about the Sierra API. After the meeting we still didn't have access to the API, but we had a better understanding of how our importer should work. Using this new understanding we began working on the front-end code for our importer script. We were able to finish the front end of the importer by having the script correctly parse the file name of the input files to our script and our script was able to correctly check the format of the file name to make sure it was a valid file using regex commands we had implemented based on the valid file name format.

With a working front end, we then created the file structure for the output of our script, which would contain the XML file containing the metadata as well as the PDF. We then used the metadata retrieval via AirPAC Classic to gain access to metadata. Using all of the metadata we had just scraped off of Addison we began working on a method to construct the XML files that would hold all of the metadata. During this stage of the project we decided to switch over from using Java to implement our script back to PHP. This switch of language will be discussed later on in the problems section of the paper. After we had finished constructing the XML files correctly using our importer script we had completed stage 3 of the project.

We took some time before moving on to stage 4 to implement error handling in our code. We moved through our previous stages of implementation to handle errors within each stage. For stage 1 we added error handling to make sure the system didn't crash anytime an invalid file name was parsed or when the regex failed. For stage 2 we handled errors for when the metadata was not retrieved or found in the Addison database. In stage 3 we handled errors for when the XML files were not created correctly. Throughout our whole implementation we also made sure to catch any system errors such as I\O errors to make sure they wouldn't crash the running of script during anyone file import. The last stage of the project was to move the XML files created in stage 3 and the PDF to the output directory. We also handled any errors within this stage of the code to make our importer script as robust as possible.

Once the first version of our final importer script was written we began implementing test cases. The tests were extensive to make sure the script would work as intended and would never crash from an error we missed during error handling. As the semester was coming to a close, we met with Jie Chen who will be maintaining our script after our completion. He provided some suggestions for the script as well as asked for our help to bring him up to speed on our implementation. We met with Jie many times in the last few weeks of the semester to polish our final script. After we had Jie's approval on our final script we finished all work on our script and had our final product by the semester's end.

Problems

When we began working on our importer script one of the very first issues we ran into was working with the old importer script. The script was poorly documented and we had to try to read through all the code and decipher exactly what was going on in the old PHP importer script, which took away from the amount of time we had to implement our own script. We also didn't have access to the original implementers of the old PHP code. Most of us were fairly new to PHP at the beginning of the project, so it took some time to learn PHP while trying to understand the old importer script. Another problem hinted at earlier in the schedule section of this report was dealing with PHP and Java. Our team originally started to implement the various pieces of our script within Java but we soon realized that translating certain methods would turn one line into 10 or much more. One key portion of our script we were having trouble implementing in Java was creating and formatting the metadata scraped off of Addison into clean XML files that were formatted into the format we had intended. At this point our team came to a decision to switch over to PHP for the implementation of the script. Another problem that we faced was that we were never given access to the Sierra. Our client, Paul Mather, originally wanted our importer script to move away from the scraping of Addison for the metadata to using the Sierra API to directly access the metadata in the database. He wanted this change because scraping the data directly off of Addison with AirPAC Classic was not the most reliable system during the importing of BTDs. One of the last issues we faced was during testing. Our team did not have access to the actual databases and system that our script would be running on. We would of liked to have access to this or a test environment similar to the original system that our importer script would be running on to help us test our script and find any errors or bugs we may have missed.

Solutions

Even though we faced some problems during the semester while implementing our importer script we usually were able to overcome these problems with some simple solutions. While the lack of documentation did effect the overall time we spent on the project, we had an extra meeting with Paul to help us try to understand what the old importer script was doing in some of the more convoluted sections. The switch from implementing our importer script in Java to implementing the script in PHP did take less time than rewriting the whole script would have taken. This saved time was due to some of PHP's built in methods that saved us more time overall and made our script simpler. Towards the end of the semester Paul let us know that he had still not gotten access to the Sierra API and that due to time constraints we should not worry about including it in our final product, so we stayed with using AirPAC Classic. We did implement our script in such a way that the use of AirPAC Classic could easily be changed out for the Sierra API when it was ready in the future. For not being able to test our importer script on the actual system it will be run on in the future, we recreated a test environment mimicking the actual environment our script would run on to the best of our knowledge. We implemented our test environment on rlogin and ran most of our tests on the rlogin cluster. These tests helped us find some bugs in our code, and the environment helped us run our test cases that we ran when we had a finished version of our importer script.

Future

Looking to the future, our importer script will eventually see the addition of the Sierra API. Once our clients have access to the new Sierra API system they may add it to our implementation to improve the reliability of our script. Another task to be completed in the future is validation of all the XML files our importer creates to make sure they are valid and match a certain schema. Most importantly, in the future our script will help in the importing of the 13,000+ BTD's. Even though our script will be improved in the future, our importer script set up the basis so BTD's can start being imported into the VTechWorks database.

Acknowledgments

First off we would like to thank our Professor Edward Fox for helping guide us through this project as worked we to meet our project requirements. Next we would like to thank our clients Paul Mather, Jie Chen, and Zhiwu Xie for helping us in understanding what they wanted out of our final product and helping us with any questions or concerns we had during the project.