Final author manuscript of: Maureen P. Walsh, "Batch Loading Collections into DSpace: Using Perl Scripts for Automation and Quality Control," *Information Technology and Libraries* 29, no. 3 (2010): 117-127.

Batch Loading Collections into DSpace: Using Perl Scripts for Automation and Quality Control

Maureen P. Walsh

Metadata Librarian / Assistant Professor
The Ohio State University Libraries, Columbus, Ohio

Abstract

This paper describes batch loading workflows developed for the Knowledge Bank, The Ohio State University's institutional repository. In the five years since the inception of the repository approximately 80 percent of the items added to the Knowledge Bank, a DSpace repository, have been batch loaded. Most of the batch loads utilized Perl scripts to automate the process of importing metadata and content files. Custom Perl scripts were used to migrate data from spreadsheets or comma-separated values files into the DSpace archive directory format, to build collections and tables of contents, and to provide data quality control. Two projects are described to illustrate the process and workflows.

Introduction

The mission of the Knowledge Bank, The Ohio State University's (OSU) institutional repository, is to collect, preserve, and distribute the digital intellectual output of OSU's faculty, staff, and students.[1] The staff working with the Knowledge Bank have sought from its inception to be as efficient as possible in adding content to DSpace. Using batch loading workflows to populate the repository has been integral to that efficiency. The first batch load into the Knowledge Bank was August 29, 2005. Over the next four years since, 698 collections containing 32,188 items were batch loaded, representing 79 percent of the items and 58 percent of the collections in the Knowledge Bank. These batch loaded collections vary from

journal issues to photo albums. The items include articles, images, abstracts, and transcripts. The majority of the batch loads, including the first, used custom Perl scripts to migrate data from Microsoft Excel spreadsheets into the DSpace batch import format for descriptive metadata and content files. Perl scripts have been used for data cleanup and quality control as part of the batch load process. Perl scripts, in combination with shell scripts, have also been used to build collections and tables of contents in the Knowledge Bank. The workflows using Perl scripts to automate batch import into DSpace have evolved through an iterative process of continual refinement and improvement. Two Knowledge Bank projects are presented as case studies to illustrate a successful approach that may be applicable to other institutional repositories.

Literature Review

Batch ingesting is acknowledged in the literature as a means of populating institutional repositories. There are examples of specific batch loading processes minimally discussed in the literature. Branschofsky and her colleagues briefly described batch loading MARC metadata crosswalked to DSpace Dublin Core (DC) in a poster session.[2] Mishra and others developed a Perl script to create the DSpace archive directory for batch import of electronic theses and dissertations (ETDs) extracted with a Java program from an in-house bibliographic database.[3] Mundle used Perl scripts to batch process ETDs for import into DSpace with MARC catalog records or Excel spreadsheets as the source metadata.[4] Brownlee used Python scripts to batch process comma-separated values (CSV) files exported from Filemaker database software for ingest via the DSpace item importer.[5]

More in-depth descriptions of batch loading are provided by Thomas; Kim, Dong, and Durden; Proudfoot et al.; Witt and Newton; Drysdale; Ribaric; Floyd; and Averkamp and Lee. However, irrespective of repository software, each describes a process to populate their repositories dissimilar to the workflows developed for the Knowledge Bank in approach or source data.

Thomas describes the Perl scripts used to convert MARC catalog records into DC and to create the archive directory for DSpace batch import.[6]

Kim, Dong, and Durden used Perl scripts to semi-automate the preparation of files for batch loading a University of Texas Harry Ransom Humanities Research Center (HRC) collection into DSpace. The XML source metadata they used was generated by the National Library of New Zealand Metadata Extraction Tool.[7] Two subsequent projects for the HRC revisited the workflow described by Kim, Dong, and Durden.[8]

Proudfoot and her colleagues discuss importing metadata-only records from departmental RefBase, Thomson Reuters EndNote, and Microsoft Access databases into ePrints. They also describe an experimental Perl script written to scrape lists of publications from personal Web sites to populate ePrints.[9]

Two additional workflow examples used citation databases as the data source for batch loading into repositories. Witt and Newton provide a tutorial on transforming EndNote metadata for Digital Commons with XSLT (Extensible Stylesheet Language Transformations).[10] Drysdale describes the Perl scripts used to convert Thomson Reuters Reference Manager files into XML for the batch loading of metadata-only records into the University of Glascow's ePrints repository.[11] The Glascow ePrints batch workflow is additionally described by Robertson and Nixon and Greig.[12]

Several workflows were designed for batch loading ETDs into repositories. Ribaric describes the automatic preparation of ETDs from the Internet Archive (http://www.archive.org/) for ingest into DSpace using PHP utilities.[13] Floyd describes the processor developed to automate the ingest of ProQuest ETDs via the DSpace item importer.[14] Also using ProQuest ETDs as the source data, Averkamp and Lee described using XSLT to transform the ProQuest data to bepress' (The Berkeley Electronic Press) schema for batch loading into a Digital Commons repository.[15]

The Knowledge Bank workflows described in this paper use Perl scripts to generate DC XML and create the archive directory for batch loading metadata records and content files into DSpace using Excel spreadsheets or CSV files as the source metadata.

Background

The Knowledge Bank, a joint initiative of the OSU Libraries (OSUL) and the OSU Office of the Chief Information Officer, was first registered in the Registry of Open Access Repositories (ROAR) on September 28, 2004.[16] As of December 2009 the repository held 40,686 items in 1,192 collections. The Knowledge Bank uses DSpace, the open-source Java-based repository software jointly developed by the Massachusetts Institute of Technology Libraries and Hewlett-Packard.[17] As a DSpace repository, the Knowledge Bank is organized by communities. The fifty-two communities currently in the Knowledge Bank include administrative units, colleges, departments, journals, library special collections, research centers, symposiums, and undergraduate honors theses. The commonality of the varied Knowledge Bank communities is their affiliation with OSU and their production of knowledge in a digital format that they wish to store, preserve, and distribute.

The staff working with the Knowledge Bank includes a team of people from three OSUL areas—Technical Services, Information Technology, and Preservation—and the contracted hours of one systems developer from the OSU Office of Information Technology (OIT). The OSUL team members are not individually assigned full-time to the repository. The current OSUL team includes a librarian repository manager, two metadata librarians, one systems librarian, one systems developer, two technical services staff members, one preservation staff member, and one graduate assistant.

The Knowledge Bank is currently running DSpace 1.5.2 and the relational database PostgreSQL 8.1.11 on the Red Hat Enterprise Linux 5 operating system. The structure of the Knowledge Bank follows the hierarchical arrangement of DSpace. Communities are at the highest level and can be divided into sub-communities. Each community or sub-community contains one or more collections. All items—the

basic archival elements in DSpace—are contained within collections. Items consist of metadata and bundles of bitstreams (files). DSpace supports two user interfaces: the original interface based on JavaServer Pages (JSPUI) and the newer Manakin (XMLUI) interface based on the Apache Cocoon framework. At this writing, the Knowledge Bank continues to use the JSPUI interface.

The default metadata used by DSpace is a Qualified DC schema derived from the DC-Library Application Profile.[18] The Knowledge Bank uses a locally defined extended version of the default DSpace Qualified DC schema, which includes several additional element qualifiers. The metadata management for the Knowledge Bank is guided by a Knowledge Bank Application Profile and a core element set for each collection within the repository derived from the Application Profile.[19] The metadata librarians at OSUL create the collection core element sets in consultation with the community representatives. The core element sets serve as metadata guidelines for submitting items into the Knowledge Bank regardless of the method of ingest.

The primary means of adding items to collections in DSpace, and the two ways used for Knowledge Bank ingest, are (1) direct (or intermediated) author entry via the DSpace Web item submission user interface and (2) in batch via the DSpace item importer. Recent enhancements to DSpace, not yet fully explored for use with the Knowledge Bank, include new ingest options using Simple Web-service Offering Repository Deposit (SWORD), Open Archives Initiative Object Reuse and Exchange (OAI-ORE), and DSpace package importers such as the Metadata Encoding and Transmission Standard Submission Information Package (METS SIP) format. This paper describes ingest via the DSpace batch item importer.

The DSpace item importer is a command-line tool for batch ingesting items. The importer uses a simple archive format diagramed in figure 1. The archive is a directory of items that contain a subdirectory of item metadata, item files, and a contents file listing the bitstream file names. Each item's descriptive metadata is contained in a DC XML file. The format used by DSpace for the DC XML files is illustrated in figure 2. Automating the process of creating the Unix archive directory has been the main

function of the Perl scripts written for the Knowledge Bank batch loading workflows. A systems developer uses the test mode of the DSpace item importer tool to validate the item directories before doing a batch load. Any significant errors are corrected and the process is repeated. After a successful test, the batch is loaded into the staging instance of the Knowledge Bank and quality checked by a metadata librarian to identify any unexpected results and script or data problems that need to be corrected. After a successful load into the staging instance the batch is loaded into the production instance of the Knowledge Bank.

Figure 1. DSpace simple archive format

```
archive_directory/
        item_000/
            dublin_core.xml -- Qualified Dublin Core metadata
            contents        -- text file containing one line per filename
            file_1.pdf      -- files to be added as bitstreams to the item
            file_2.pdf
        item_001/
            dublin_core.xml
            contents
            file_1.pdf
            . . .
```

Figure 2. DSpace Qualified Dublin Core XML

```
<dublin_core>
        <dcvalue element="title" qualifier="none">Notes on the Bird Life of Cedar Point</dcvalue>
        <dcvalue element="date" qualifier="issued">1901-04</dcvalue>
        <dcvalue element="creator" qualifier="none">Griggs, Robert F.</dcvalue>
</dublin_core>
```

Most of the Knowledge Bank batch loading workflows use Excel spreadsheets or CSV files as the source for the descriptive item metadata. The creation of the metadata contained in the spreadsheets or files has varied by project. In some cases the metadata is created by OSUL staff. In other cases the metadata is supplied by Knowledge Bank communities in consultation with a metadata librarian or by a vendor contracted by OSUL. Whether the source metadata is created in-house or externally supplied, OSUL staff are involved in the quality control of the metadata.

Several of the first communities to join the Knowledge Bank had very large retrospective collection sets to archive. The collection sets of two of those early adopters, the journal issues of *The Ohio Journal of Science* (*OJS*) and the abstracts of the OSU International Symposium on Molecular Spectroscopy currently account for 59 percent of the items in the Knowledge Bank.[20] The successful batch loading workflows developed for these two communities—which continue to be active content suppliers to the repository—are presented as case studies.

Case Studies

The Issues of *The Ohio Journal of Science*

OJS was jointly published by OSU and The Ohio Academy of Science (OAS) until 1974, when OAS took over sole control of the journal. The issues of *OJS* are archived in the Knowledge Bank with a two year rolling wall embargo. The issues for 1900 through 2003, a total of 639 issues containing 6,429 articles, were batch loaded into the Knowledge Bank. Due to rights issues, the retrospective batch loading project had two phases. The project to digitize *OJS* began with the 1900-1972 issues that OSU had the rights to digitize and make publically available. OSU later acquired the rights for 1973-present, and (accounting for the embargo period) 1973-2003 became phase 2 of the project. The two phases of batch loads were the most complicated automated batch loading processes developed to date for the Knowledge Bank. To batch load phase 1 in 2005 and phase 2 in 2006, the systems developers working with the Knowledge

Bank wrote scripts to build collections, generate DC XML from the source metadata, create the archive directory, load the metadata and content files, create tables of contents, and load the tables of contents into DSpace.

The *OJS* community in the Knowledge Bank is organized by collections representing each issue of the journal. The systems developers used scripts to automate the building of the collections in DSpace because of the number needed as part of the retrospective project. The individual articles within the issues are items within the collections. There is a table of contents for the articles in each issue as part of the collection homepages.[21] Again, due to the number required for the retrospective project, the systems developers used scripts to automate the creation and loading of the tables of contents. The tables of contents are contained in the HTML introductory text section of the collection pages. The tables of contents list title, authors, and pages. They also include a link to the item record and a direct link to the article PDF which includes the file size.

For each phase of the *OJS* project, a vendor contracted by OSUL supplied the article PDFs and an Excel spreadsheet with the article-level metadata. The metadata received from the vendor had not been customized for the Knowledge Bank. The *OJS* issues were sent to a vendor for digitization and metadata creation before the Knowledge Bank was chosen as the hosting site of the digitized journal. The OSU Digital Initiatives Steering Committee 2002 proposal for the *OJS* digitization project had predated the Knowledge Bank DSpace instance. OSUL staff performed quality control checks of the vendor-supplied metadata and standardized the author names. The vendor supplied the author names as they appeared in the articles—in direct order, comma separated, and including any "and" that appeared. In addition to other quality checks performed, OSUL staff edited the author names in the spreadsheet to conform to DSpace author-entry convention (surname first). Semicolons were added to separate author names and the extraneous ands were removed. A former metadata librarian mapped the vendor-supplied article-level metadata to Knowledge Bank DC, as illustrated in table 1. The systems developers used the mapping as a guide to write Perl scripts to transform the vendor metadata into the DSpace schema of DC.

Table 1. Mapping of vendor metadata to Qualified Dublin Core

Vendor-Supplied Metadata	Knowledge Bank Dublin Core
File	[n/a: PDF file name]
Cover Title	dc.identifier.citation*
ISSN	dc.identifier.issn
Vol.	dc.identifier.citation*
Iss.	dc.identifier.citation*
Cover Date	dc.identifier.citation*
Year	dc.date.issued
Month	dc.date.issued
Fpage	dc.identifier.citation*
Lpage	dc.identifier.citation*
Article Title	dc.title
Author Names	dc.creator
Institution	dc.description
Abstract	dc.description.abstract
n/a	dc.language.iso
n/a	dc.rights
n/a	dc.type

*format: [Cover Title]. v[Vol.], n[Iss.] ([Cover Date]), [Fpage]-[Lpage]

The workflow for the two phases was nearly identical, except each phase had its own batch loading scripts. Due to a staff change between the two phases of the project, a former OSUL systems developer was responsible for batch loading phase 1 and the OIT systems developer was responsible for phase 2. The phase 1 scripts were all written in Perl. The four scripts written for phase 1 created the archive directory, performed database operations to build the collections, generated the HTML introduction table of contents for each collection, and loaded the tables of contents into DSpace via the

database. For phase 2, the OIT systems developer modified and added to the phase 1 batch processing scripts. This case study focuses on phase 2 of the project.

Batch Processing for Phase 2 of *OJS*

The annotated scripts the OIT systems developer used for phase 2 of the *OJS* project are included in appendix A. A shell script (mkcol.sh) added collections based on a listing of the journal issues. The script performed a login as a selected user ID to the DSpace Web interface using the Web access tool Curl. A subsequent simple looping Perl script (mkallcol.pl) used the stored credentials to submit data via this channel to build the collections in the Knowledge Bank.

The metadata.pl script created the archive directory for each collection. The OIT systems developer added the PDF file for each item to Unix. The vendor-supplied metadata was saved as Unicode text format and transferred to Unix for further processing. The developer used vi commands to manually modify metadata for characters illegal in XML (e.g., "<" and "&"). (Although manual steps were used for this project, the OIT systems developer improved the Perl scripts for subsequent projects by adding code for automated transformation of the input data to help ensure XML validity.) The metadata.pl script then processed each line of the metadata along with the corresponding data file. For each item, the script created the DC XML file and the contents file and moved them and the PDF file to the proper directory. Load sets for each collection (issue) were placed in their own subdirectory, and a load was done for each subdirectory. The items for each collection were loaded by a small Perl script (loaditems.pl) that used the list of issues and their collection IDs and called a shell script (import.sh) for the actual load.

The tables of contents for the issues were added to the Knowledge Bank after the items were loaded. A Perl script (intro.pl) created the tables of contents using the metadata and the DSpace map file, a stored mapping of item directories to item handles created during the load. The tables of contents were added to the Knowledge Bank using a shell script (installintro.sh) similar to what was used to create the collections. Installintro.sh used Curl to simulate a user adding the data to DSpace by performing a login

as a selected user ID to the DSpace Web interface. A simple looping Perl script (ldallintro.pl) called installintro.sh and used the stored credentials to submit the data for the tables of contents.

The Abstracts of the OSU International Symposium on Molecular Spectroscopy

The Knowledge Bank contains the abstracts of the papers presented at the OSU International Symposium on Molecular Spectroscopy (MSS), which has met annually since 1946. Beginning with the 2005 Symposium, the complete presentations from authors who have authorized their inclusion are archived along with the abstracts. The MSS community in the Knowledge Bank currently contains 17,714 items grouped by decade into six collections. The six collections were created "manually" via the DSpace Web interface prior to the batch loading of the items. The retrospective years of the Symposium (1946-2004) were batch loaded in three phases in 2006. Each Symposium year following the retrospective loads was batch loaded individually.

Retrospective MSS Batch Loads

The majority of the abstracts for the retrospective loads were digitized by OSUL. A vendor was contracted by OSUL to digitize the remainder and to supply the metadata for the retrospective batch loads. The files digitized by OSUL were sent to the vendor for metadata capture. OSUL provided the vendor a metadata template derived from the MSS core element set. The metadata taken from the abstracts comprised author, affiliation, title, year, session number, sponsorship (if applicable), and a full transcription of the abstract. To facilitate searching, the formulas and special characters appearing in the titles and abstracts were encoded using LaTeX, a document preparation system used for scientific data. The vendor delivered the metadata in Excel spreadsheets as per the spreadsheet template provided by OSUL. Quality checking the metadata was an essential step in the workflow for OSUL. The metadata received for the project required revisions and data cleanup. The vendor originally supplied incomplete

files and spreadsheets that contained data errors, including incorrect numbering, data in the wrong fields, and inconsistency with the LaTeX encoding.

The three Knowledge Bank batch load phases for the retrospective MSS project corresponded to the staged receipt of metadata and digitized files from the vendor. The annotated scripts used for phase 2 of the project, which included twenty years of the Symposium between 1951 and 1999, are included in appendix B. The OIT systems developer saved the metadata as a tab-separated file and added it to Unix along with the abstract files. A Perl script (mkxml2.pl) transformed the metadata into DC XML and created the archive directories for loading the metadata and abstract files into the Knowledge Bank. The script divided the directories into separate load sets for each of the six collections and accounted for the inconsistent naming of the abstract files. The script added the constant data for type and language that was not included in the vendor-supplied metadata. Unlike the *OJS* project, where multiple authors were on the same line of the metadata file, the MSS phase 2 script had to code for authors and their affiliations on separate lines. Once the load sets were made, the OIT systems developer ran a shell script to load them. The script (import_collections.sh) was used to run the load for each set so that the DSpace item import command did not need to be constructed each time.

Annual MSS Batch Loads

A new workflow was developed for batch loading the annual MSS collection additions. The metadata and item files for the annual collection additions are supplied by the MSS community. The community provides the Symposium metadata in a CSV file and the item files in a Tar archive file. The Symposium uses a Web form for LaTex-formatted abstract submissions. The community processes the electronic Symposium submissions with a Perl script to create the CSV file. The metadata delivered in the CSV file is based on the template created by the author, which details the metadata requirements for the project.

The OIT systems developer borrowed from and modified earlier Perl scripts to create a new script for batch processing the metadata and files for the annual Symposium collection additions. To assist with

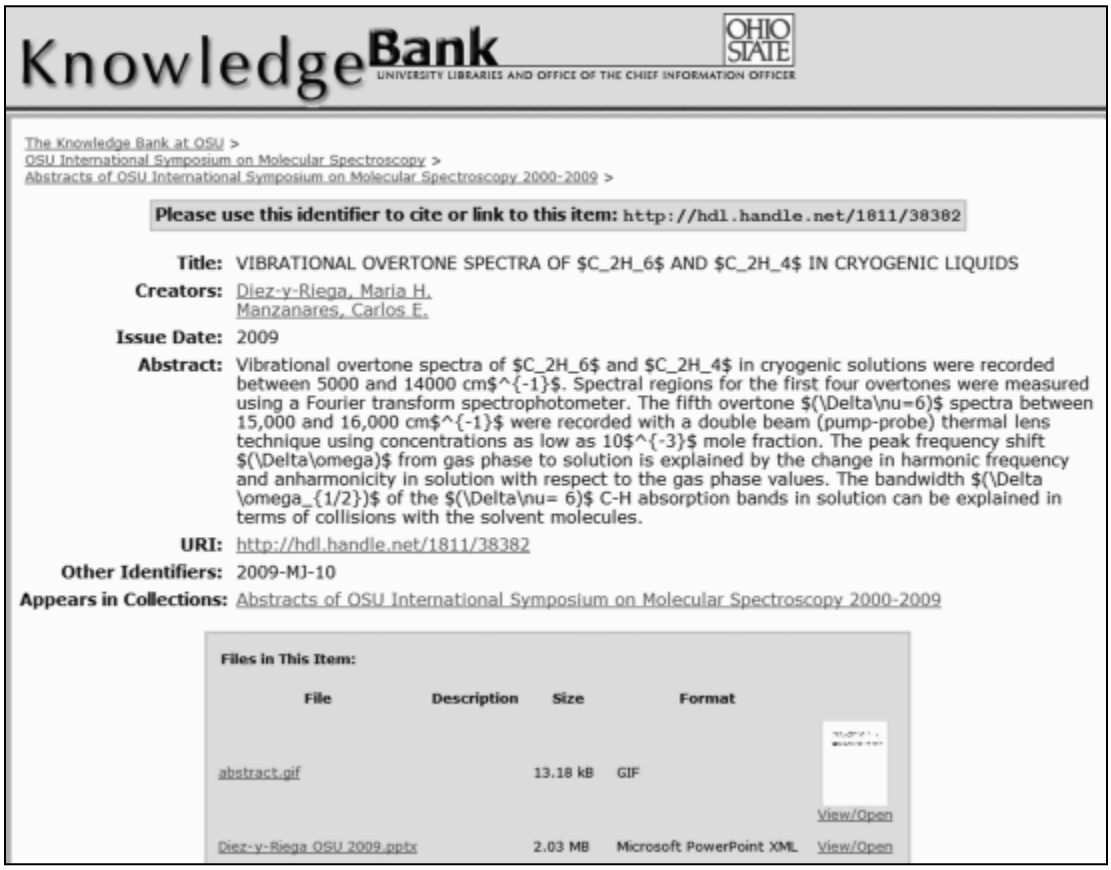
the development of the new script, I provided the developer a mapping of the community CSV headings to the Knowledge Bank DC fields. I also provided a sample DC XML file to illustrate the desired result of the Perl transformation of the community metadata into DC XML. For each new year of the Symposium, I create a sample DC XML result for an item to check the accuracy of the script. A DC XML example from a 2009 MSS item is included in appendix C. Unlike the previous retrospective MSS loads in which the script processed multiple years of the Symposium, the new script processes one year at a time. The annual Symposiums are batch loaded individually into one existing MSS decade collection. The new script for the annual loads was tested and refined by loading the 2005 Symposium into the staging instance of the Knowledge Bank. Problems encountered with character encoding and file types were resolved by modifying the script.

The metadata and files for the Symposium years 2005, 2006, and 2007 were made available to OSUL in 2007, and each year was individually loaded into the existing Knowledge Bank collection for that decade. These first three years of community-supplied CSV files contained author metadata inconsistent with Knowledge Bank author entries. The names were in direct order, uppercase, split by either a semicolon or "and", and included extraneous data, such as an address. The OIT systems developer wrote a Perl script to correct the author metadata as part of the batch loading workflow. An annotated section of that script illustrating the author modifications is included in appendix D. The MSS community revised the Perl script they used to generate the CSV files by including an edited version of this author entry correction script and were able to provide the expected author data for 2008 and 2009. The author entries received for these years were in inverted order (surname first) and mixed case, were semicolon separated, and included no extraneous data. The receipt of consistent data from the community for the last two years has facilitated the standardized workflow for the annual MSS loads.

The scripts used to batch load the 2009 Symposium year are included in appendix E. The OIT systems developer unpacked the Tar file of abstracts and presentations into a directory named for the year of the Symposium on Unix. The Perl script written for the annual MSS loads

(mkxml<year>.pl) was saved on Unix and renamed mkxml2009.pl. The script was edited for 2009 (including the name of the CSV file and the location of the directories for the unpacked files and generated XML). The CSV headings used by the community in the new file were checked and verified against the extract list in the script. Once the Perl script was up-to-date and the base directory was created, the OIT systems developer ran the Perl script to generate the archive directory set for import. The import.sh script was then edited for 2009 and run to import the new Symposium year into the staging instance of the Knowledge Bank as a quality check prior to loading into the live repository. The brief item view of an example MSS 2009 item archived in the Knowledge Bank is shown in figure 3.

Figure 3. MSS 2009 archived item example



Summary and Conclusion

Each of the batch loads that used Perl scripts had its own unique features. The format of content and associated metadata varied considerably, and custom scripts to convert the content and metadata into the DSpace import format were created on a case-by-case basis. The differences between batch loads included the delivery format of the metadata, the fields of metadata supplied, how metadata values were delimited, the character set used for the metadata, the data used to uniquely identify the files to be loaded, and how repeating metadata fields were identified. Because of the differences in supplied metadata, a separate Perl script for generating the DC XML and archive directory for batch loading was written for each project. Each new Perl script borrowed from and modified earlier scripts. Many of the early batch loads were firsts for the Knowledge Bank and the staff working with the repository, both in terms of content and in terms of metadata. Dealing with community- and vendor-supplied metadata and various encodings (including LaTeX), each of the early loads encountered different data obstacles, and in each case solutions were written in Perl. The batch loading code has matured over time, and the progression of improvements is evident in the example scripts included in the appendixes.

Batch loading can greatly reduce the time it takes to add content and metadata to a repository, but successful batch loading workflows are dependent upon the quality of data and metadata loaded. Along with testing scripts and checking imported metadata by first batch loading to a development or staging environment, quality control of the supplied metadata is an integral step. The flexibility of Perl allowed testing and revising to accommodate problems encountered with how the metadata was supplied for the heterogeneous collections batch loaded into the Knowledge Bank. However, toward the goal of standardizing batch loading workflows, the staff working with the Knowledge Bank iteratively refined not only the scripts but also the metadata requirements for each project and how those were communicated to the data suppliers with mappings, explicit metadata examples, and sample desired results. The efficiency of batch loading workflows is greatly enhanced by consistent data and basic standards for how metadata is supplied.

Batch loading is not only an extremely efficient means of populating an institutional repository, it is also a value-added service that can increase buy-in from the wider campus community. It is hoped that by openly sharing examples of our batch loading scripts we are contributing to the development of an open library of code that can be borrowed and adapted by the library community toward future institutional repository success stories.

Acknowledgements

I would like to thank Conrad Gratz, of OSU OIT, and Andrew Wang, formerly of OSUL. Gratz wrote the shell scripts and the majority of the Perl scripts used for automating the Knowledge Bank item import process and ran the corresponding batch loads. The early Perl scripts used for batch loading into the Knowledge Bank, including the first phase of *OJS* and MSS, were written by Wang. Parts of those early Perl scripts written by Wang were borrowed for subsequent scripts written by Gratz. Gratz provided the annotated scripts appearing in the appendixes and consulted with the author regarding the description of the scripts. I would also like to thank Amanda J. Wilson, a former metadata librarian for OSUL, who was instrumental to the success of many of the batch loading workflows created for the Knowledge Bank.

References and Notes

[1] The Ohio State University Knowledge Bank, "Institutional Repository Policies," 2007, http://library.osu.edu/sites/kbinfo/policies.html (accessed Dec. 21, 2009). The Knowledge Bank homepage can be found at https://kb.osu.edu/dspace/ (accessed Dec. 21, 2009).

[2] Margret Branschofsky et al, "Evolving Metadata Needs for an Institutional Repository: MIT's DSpace," Proceedings of the 2003 International Conference on Dublin Core and Metadata Applications: Supporting Communities of Discourse and Practice—Metadata Research & Applications, Seattle, Washington, 2003, http://dcpapers.dublincore.org/ojs/pubs/article/view/753/749 (accessed Dec. 21, 2009).

[3] R. Mishra et al, "Development of ETD Repository at IITK Library using DSpace," in *International Conference on Semantic Web and Digital Libraries (ICSD-2007)*, ed. A. R. D. Prasad and Devika P. Madalli (2007), 249-259. http://hdl.handle.net/1849/321 (accessed Dec. 21, 2009).

[4] Todd M. Mundle, "Digital Retrospective Conversion of Theses and Dissertations: An In House Project" (paper presented to the 8th International Symposium on Electronic Theses & Dissertations, Sydney, Australia, September 28-30, 2005), http://adt.caul.edu.au/etd2005/papers/080Mundle.pdf (accessed Dec. 21, 2009).

[5] Rowan Brownlee, "Research Data and Repository Metadata: Policy and Technical Issues at the University of Sydney Library," *Cataloging & Classification Quarterly* 47, no. 3/4 (2009): 370-379.

[6] Steve Thomas, “Importing MARC Data into DSpace,” 2006, http://hdl.handle.net/2440/14784 (accessed Dec. 21, 2009).

[7] Sarah Kim, Lorraine A. Dong, and Megan Durden, "Automated Batch Archival Processing: Preserving Arnold Wesker's Digital Manuscripts," *Archival Issues* 30, no. 2 (2006): 91-106.

[8] Elspeth Healey, Samantha Mueller, and Sarah Ticer, “The Paul N. Banks Papers: Archiving the Electronic Records of a Digitally-Adventurous Conservator,” 2009, https://pacer.ischool.utexas.edu/bitstream/2081/20150/1/Paul_Banks_Final_Report.pdf (accessed Dec. 21, 2009); Lisa Schmidt, “Preservation of a Born Digital Literary Genre: Archiving Legacy Macintosh Hypertext Files in DSpace,” 2007, https://pacer.ischool.utexas.edu/bitstream/2081/9007/1/MJ%20WBO%20Capstone%20Report.pdf (accessed Dec. 21, 2009).

[9] Rachel E. Proudfoot et al, "JISC Final Report: IncReASe (Increasing Repository Content through Automation and Services)," 2009, http://eprints.whiterose.ac.uk/9160/ (accessed Dec. 21, 2009).

[10] Michael Witt and Mark P. Newton, "Preparing Batch Deposits for Digital Commons Repositories," 2008, http://docs.lib.purdue.edu/lib_research/96/ (accessed Dec. 21, 2009).

---

[11] Lesley Drysdale, "Importing Records from Reference Manager into GNU EPrints," 2004, http://hdl.handle.net/1905/175 (accessed Dec. 21, 2009).

[12] R. John Robertson, "Evaluation of Metadata Workflows for the Glasgow ePrints and DSpace Services," 2006, http://hdl.handle.net/1905/615 (accessed Dec. 21, 2009); William J. Nixon and Morag Greig, "Populating the Glasgow ePrints Service: A Mediated Model and Workflow," 2005, http://hdl.handle.net/1905/387 (accessed Dec. 21, 2009).

[13] Tim Ribaric, "Automatic Preparation of ETD Material from the Internet Archive for the DSpace Repository Platform," *Code4Lib Journal*, no. 8 (Nov. 23, 2009), http://journal.code4lib.org/articles/2152 (accessed Dec. 21, 2009).

[14] Randall Floyd, “Automated Electronic Thesis and Dissertations Ingest,” March 30, 2009, http://wiki.dlib.indiana.edu/confluence/x/01Y (accessed Dec. 21, 2009).

[15] Shawn Averkamp and Joanna Lee, "Repurposing ProQuest Metadata for Batch Ingesting ETDs into an Institutional Repository," *Code4Lib Journal*, no. 7 (June 26, 2009), http://journal.code4lib.org/articles/1647 (accessed Dec. 21, 2009).

[16] Tim Brody, Registry of Open Access Repositories (ROAR), http://roar.eprints.org/ (accessed Dec. 21, 2009).

[17] DuraSpace, DSpace, http://www.dspace.org/ (accessed Dec. 21, 2009).

[18] Dublin Core Metadata Initiative Libraries Working Group, “DC-Library Application Profile (DC-Lib),” http://dublincore.org/documents/2004/09/10/library-application-profile/ (accessed Dec. 21, 2009).

[19] The Ohio State University Knowledge Bank Policy Committee, “OSU Knowledge Bank Metadata Application Profile,” http://library.osu.edu/sites/techservices/KBAppProfile.php (accessed Dec. 21, 2009).

[20] Ohio Journal of Science (Ohio Academy of Science), Knowledge Bank community, http://hdl.handle.net/1811/686 (accessed Dec. 21, 2009); OSU International Symposium on Molecular Spectroscopy, Knowledge Bank community, http://hdl.handle.net/1811/5850 (accessed Dec. 21, 2009).

[21] Ohio Journal of Science (Ohio Academy of Science), Ohio Journal of Science: Volume 74, Issue 3 (May, 1974), Knowledge Bank collection, http://hdl.handle.net/1811/22017 (accessed Dec. 21, 2009).

Appendix A. *OJS* Batch Loading Scripts

-- mkcol.sh --

```
#!/bin/sh
# Create a Collection given a name and a collection handle.
# Gets information from DSpace web pages and returns data via GET parameters to the DSpace
# Collection Wizard.

NAME="$1"
COLLECTION_HANDLE="$2"

URL="https://kb.osu.edu/dspace"
NAME_PAT=">$NAME</option>"

# Login to DSpace and create the cookie.txt file.
curl -k -L -s $URL/password-login -d "login_email=[name removed]@osu.edu" -d
"login_password=XXXXX" -c cookie.txt > /dev/null

# Cut the community_id out of the web page.
COMMUNITY_ID=`curl -k -L -s -b cookie.txt \
  $URL/handle/1811/$COLLECTION_HANDLE \
  | grep -m1 name=\"community_id\" \
  | cut -d\" -f6`

# Cut the collection_id out of the web page.
COLLECTION_ID=`curl -k -L -s -b cookie.txt \
  $URL/tools/collection-wizard \
  -d "community_id=$COMMUNITY_ID" \
  | grep -m1 name=\"collection_id\" \
  | cut -d\" -f6`

# Begin building the collection.
curl -k -L -s -b cookie.txt \
  $URL/tools/collection-wizard \
  -d "public_read=true" \
  -d "workflow1=true" \
  -d "workflow2=" \
  -d "workflow3=" \
  -d "collection_id=$COLLECTION_ID" \
  -d "default-item=" \
```

```
   -d "stage=1" \
   -d "admins=" > /dev/null

# Finish making the collection.
curl -k -L -s -b cookie.txt \
   $URL/tools/collection-wizard \
   -F "name=$NAME" \
   -F "short_description=" \
   -F "introductory_text=" \
   -F "copyright_text=" \
   -F "side_bar_text=" \
   -F "provenance_description=" \
   -F "license=" \
   -F "file=" \
   -F "collection_id=$COLLECTION_ID" \
   -F "stage=2" \
   -F "permission=12"  > /dev/null

# Get and return the handle_id.
HANDLE_ID=`curl -k -L -s -b cookie.txt \
   $URL/handle/1811/$COLLECTION_HANDLE \
   | grep -m1 "$NAME_PAT" \
   | cut -d\" -f2`
echo $HANDLE_ID
```

-- mkallcol.pl --

```
#!/usr/bin/perl

# Routine to clean up individual fields.
sub trim($)
{
    my $string = shift;
    $string =~ s/^\s+//;
    $string =~ s/\s+$//;
    return $string;
}

# Read the file of issue names into an array.
open(fh,"issues-prod.remainder");
@lines=<fh>;
close(fh);

$linenum = 0;
%lt=();

$COMMUNITY = "686";

# For each issue get the parameters from the array and call the script to create the collection.
while ($linenum <= $#lines) {
        @fields = split(/\t/, $lines[$linenum]);
        $issue = $fields[1];
        chop($issue);
```

```
        system("echo -n $fields[0] ");
        print " ";
        system("./mkcol.sh $issue $COMMUNITY");
        $linenum++;
}
```

-- Sample of the file of issue names --

```
V074N2  "Ohio Journal of Science: Volume  74, Issue 2 (March, 1974)"
V074N3  "Ohio Journal of Science: Volume  74, Issue 3 (May, 1974)"
V074N4  "Ohio Journal of Science: Volume  74, Issue 4 (July, 1974)"
V074N5  "Ohio Journal of Science: Volume  74, Issue 5 (September, 1974)"
```

-- metadata.pl --

```
#!/usr/bin/perl

use Encode;      # Routines for UTF encoding.

# Routine to clean up individual fields of metadata.
sub trim($)
{
    my $string = shift;
    $string =~ s/^\s+//;
    $string =~ s/\s+$//;
    return $string;
}

# Read the metadata into an array.
open(fh,"<:encoding(UTF-16)", "OJSPhase2-1.txt");
@lines=<fh>;
close(fh);

# Process each line of metadata, consolidating lines for the same item.
$linenum = 0;
%lt=();
while ($linenum <= $#lines) {
        @fields = split(/\t/, $lines[$linenum]);
        if ($fields[0] =~ /^((v|V)[0-9]+(n|N)[0-9A-Za-z]+)/) {
                $lt{uc($1)} = [@{$lt{uc($1)}}, $linenum];
        }
        $linenum++;
}

# Build the load set for each item.
for $key (sort(keys(%lt))) {
        # Put each load set in its own subdirectory.
        print "mkdir ./src/$key\n";
        system "mkdir ./src/$key";
        # Process the lines for this load set.
        for $i (0 .. $#{$lt{$key}}) {
                $dir = sprintf("item_%03d", $i);
                print "mkdir ./src/$key/$dir\n";
```

```
            system "mkdir ./src/$key/$dir";
            # Create the XML for the metadata.
            open(fh,">:encoding(UTF-8)", "./src/$key/$dir/dublin_core.xml");
            print fh '<dublin_core>'."\n";
            @fields = split(/\t/, $lines[$lt{$key}[$i]]);
            $fields[1] =~ s/"//g;
            $fields[5] =~ s/"//g;
            if (length($fields[9])>0) {
               print fh '<dcvalue element="identifier" qualifier="citation">'
                     . "$fields[1]. v$fields[3], n$fields[4] ($fields[5]), $fields[8]-$fields[9]</dcvalue>\n";
            } else {
               print fh '<dcvalue element="identifier" qualifier="citation">'
                    ."$fields[1]. v$fields[3], n$fields[4] ($fields[5]), $fields[8]</dcvalue>\n";
            }
            if (length($fields[10]) > 0) {
                  $fields[10] =~ s/["]{1}([^"])/$1/g;
                  $fields[10] =~ s/("|"")$//g;
                  print fh '<dcvalue element="title" qualifier="">'.$fields[10]."</dcvalue>\n";
            }
            print fh '<dcvalue element="identifier" qualifier="issn">'.$fields[2]."</dcvalue>\n";
            print fh '<dcvalue element="date" qualifier="issued">'.$fields[6]."-".$fields[7]."</dcvalue>\n";
             # Process multiple authors.
            if (length($fields[11]) > 0) {
                  $fields[11] =~ s/"//g;
                  @authors = split(/;/,$fields[11]);
                  foreach $author (@authors) {
                        $author =~ s/^\s+//;
                        if (length($author) > 0) {
                              print fh '<dcvalue element="creator" qualifier="">'.$author.'</dcvalue>'."\n";
                        }
                  }
            }
            if (length($fields[12]) > 0) {
                  $fields[12] =~ s/"//g;
                  print fh '<dcvalue element="description" qualifier="">Author Institution:
'.$fields[12]."</dcvalue>\n";
            }
            if (length($fields[13]) > 0) {
                  $fields[13] =~ s/"//g;
                  print fh '<dcvalue element="description" qualifier="abstract">'.$fields[13]."</dcvalue>\n";
            }
            print fh "</dublin_core>\n";
            close(fh); # Finished creating the XML file.

             # Create the contents file.
            open(fh, ">./src/$key/$dir/contents");
            $fields[0] = trim($fields[0]);
            print fh "$fields[0].pdf\n";
            close(fh);

             # Move the data files into the load set.
            print "cp pdfs/$fields[0] ./src/$key/$dir\n";
            system "cp pdfs/$fields[0].pdf ./src/$key/$dir";
      }
}
```

-- loaditems.pl --

```
#!/usr/bin/perl

#Load the list of issues into an array.
open(fh,"loaditems");
@lines=<fh>;
close(fh);

# Process each issue.
$linenum = 0;
while ($linenum <= $#lines) {
	@fields = split(/ /, $lines[$linenum]);
	chop($fields[1]);
	# Add the issue to DSpace.
	system("./import.sh $fields[1] $fields[0]");
	$linenum++;
}
```

-- Sample of the load items file --

```
V074N2 1811/22016
V074N3 1811/22017
V074N4 1811/22018
V074N5 1811/22019
```

-- import.sh --

```
#!/bin/sh

# import.sh collection_id dir
# Import a collection from files generated on dspace
# Requires the directory of the destination collection and the collection id.

COLLECTION_ID=$1
EPERSON=[name removed]@osu.edu
SOURCE_DIR=./src/$2
MAP_DIR=./prod-map/
BASE_ID=`basename $COLLECTION_ID`
MAPFILE=./$MAP_DIR/map.$2
/dspace/bin/dsrun org.dspace.app.itemimport.ItemImport --add --eperson=$EPERSON
--collection=$COLLECTION_ID --source=$SOURCE_DIR --mapfile=$MAPFILE
```

-- intro.pl --

```
#!/usr/bin/perl

# Routine to clean up individual fields.
```

```
sub trim($)
{
    my $string = shift;
    $string =~ s/^\s+//;
    $string =~ s/\s+$//;
    return $string;
}

# Read the metadata into an array.
open(fh,"<:encoding(UTF-16)", "OJSPhase2-1.txt")
    or die "Can't open metadata file: $!";
@lines=<fh>;
close(fh);

# Process each line of metadata, consolidating lines for the same item.
$linenum = 0;
%lt=();
while ($linenum <= $#lines) {
        @fields = split(/\t/, $lines[$linenum]);
        if ($fields[0] =~ /^((v|V)[0-9]+(n|N)[0-9A-Za-z]+)/) {
                $lt{uc($1)} = [@{$lt{uc($1)}}, $linenum];
        }
        $linenum++;
}

# Assemble each intro.
for $key (sort(keys(%lt))) {
        open(fh,"./prod-map/map.$key") or next;
        @fids=<fh>;
        close(fh);
        @fids = sort(@fids);

        print "Generating intro for $key ...\n";
        open(fh,">:encoding(UTF-8)", "./src/$key/intro");

        # Create the HTML for each article.
        for ($i = 0; $i <= $#{$lt{$key}}; $i++) {
                @fields = split(/\t/, $lines[$lt{$key}[$i]]);
                if (length($fields[10]) > 0) {
                        $fields[10] =~ s/["]{1}([^"])/$1/g;
                        $fields[10] =~ s/("|"")$//g;
                        print fh "<strong>$fields[10]</strong><br>\n";
                }
                # Create the list of authors.
                $authcnt = 0;
                if (length($fields[11]) > 0) {
                        $fields[11] =~ s/"//g;
                        @authors = split(/;/,$fields[11]);
                        foreach $author (@authors) {
                                $author =~ s/^\s+//;
                                if ($authcnt > 0) {
                                        print fh "; $author";
                                } else {
                                        print fh $author;
                                }
```

```
                        $authcnt++;
                    }
                }
                # Add page numbers.
                if (length($fields[8]) > 0) {
                    print fh " pp. $fields[8]";
                }
                if (length($fields[9]) > 0) {
                    print fh "-$fields[9]";
                }
                print fh "<br>\n";
                # Create links for each article.
                @item_hid = split(/\s/,$fids[$i]);
                $itemno = $item_hid[0];
                $itemhid = $item_hid[1];
                $fields[0] = trim($fields[0]);
                $filename = "./src/$key/$itemno/".$fields[0].".pdf";
                @st = stat($filename) or die "No $filename: $!";
                $size = int($st[7]/1024);
                $url_1 = "/dspace/handle/$itemhid";
                $url_2 = "/dspace/bitstream/$itemhid/1/$fields[0]";
                print fh '<a href="'.$url_1.'">Article description</a> | <a href="'.$url_2.'">Article Full Text PDF
('.$size.'KB)</a><br><br>';
                print fh "\n";
            }
            close(fh);
}
```

-- installintro.sh --

```
#!/bin/sh
# Install an intro given a dir and a community id.
DIR="$1"
HANDLE="$2"
URL="https://kb.osu.edu/dspace"
# Login to DSpace
curl -k -L -s $URL/password-login -d "login_email=[name removed]@osu.edu" -d
"login_password=password" -c cookie.txt > /dev/null

# Cut the community_id out of the web page.
COMMUNITY_ID=`curl -k -L -s -b cookie.txt \
   $URL/handle/$HANDLE \
   | grep -m1 name=\"community_id\" \
   | cut -d\" -f6`

# Cut the collection_id out of the web page.
COLLECTION_ID=`curl -k -L -s -b cookie.txt \
   $URL/handle/$HANDLE \
   | grep -m1 name=\"collection_id\" \
   | cut -d\" -f6`

# Cut the title out of the web page.
TITLE=`curl -k -L -s -b cookie.txt \
```

```
    $URL/tools/edit-communities \
    -d "community_id=$COMMUNITY_ID" \
    -d "collection_id=$COLLECTION_ID" \
    -d "action=4" \
    | grep -m1 name=\"name\" \
    | cut -d\" -f6`

# Put the introductory text in DSpace.
curl -k -L -s -b cookie.txt \
    $URL/tools/edit-communities \
    -d "name=$TITLE" \
    -d "short_description=" \
    -d "introductory_text=`cat ./src/$DIR/intro`" \
    -d "copyright_text=" \
    -d "side_bar_text=" \
    -d "license=" \
    -d "provenance_description=" \
    -d "community_id=$COMMUNITY_ID" \
    -d "collection_id=$COLLECTION_ID" \
    -d "create=false" \
    -d "action=9" \
    -d "submit=Update" > /dev/null
```

-- ldallintro.pl --

```
#!/usr/bin/perl

# Load file of issues into an array.
open(fh,"loaditems");
@lines=<fh>;
close(fh);

$linenum = 0;
%lt=();

# Process each intro.
while ($linenum <= $#lines) {
        @fields = split(/\t/, $lines[$linenum]);
        print("$lines[$linenum]");
        system("./installintro.sh $lines[$linenum] ");
        $linenum++;
}
```

Appendix B. MSS Phase 2 Scripts

-- mkxml2.pl --

```
#!/usr/bin/perl

# Load routines for UTF-16 and UTF-8
use Encode;

# Routine to clean up metadata fields
sub trim($)
{
   my $string = shift;
   $string =~ s/^\s+//;
   $string =~ s/\s+$//;
   $string =~ s/^"//;
   $string =~ s/"$//;
   return $string;
}

# Load metadata into an array.
open(fh,"<:encoding(UTF-16)", "MSA-phase-2-v3.txt");
@lines=<fh>;
close(fh);

$linenum = 0;
%lt=();

# Split tab separated metadata fields
while ($linenum <= $#lines) {
        @fields = split(/\t/, $lines[$linenum]);
        if ($fields[4] =~ /^([0-9]{4}-[^0-9]+[0-9]+)/) {
                $lt{$1} = [@{$lt{$1}}, $linenum];
        }
        $linenum++;
}

$cnt1 = 0; $cnt2 = 0; $cnt3 = 0; $cnt4 = 0; $cnt5 = 0; $cnt6 = 0;

# Process metadata line by line
for $key (sort(keys(%lt))) {
      $year =  substr($key, 0, 4);

        # Generate possible image file names.
        $keyzero =  substr($key,0,-1). "0" . substr($key, -1, 1);
        $keyuc =  uc($key);
        $keyuczero =  uc($keyzero);

        # Compensate for inconsistent naming of images in metadata.
      if (-e "../images/$year/$key.jpg") {
          $filename = $key;
```

```
} elsif (-e "../images/$year/$keyzero.jpg") {
        $filename = $keyzero;
} elsif (-e "../images/$year/$keyuc.jpg") {
        $filename = $keyuc;
} elsif (-e "../images/$year/$keyuczero.jpg") {
        $filename = $keyuczero;
} else {
                $filename = "";
        print " NO FILE FOUND images/$year/$key.jpg\n";
    }

    # Divide output into separate load sets based on year.
if (($year >= "1946") && ($year <= "1959")) {
    $dir = sprintf("1/item_%04d", $cnt1++);
}
if (($year >= "1960") && ($year <= "1969")) {
    $dir = sprintf("2/item_%04d", $cnt2++);
}
if (($year >= "1970") && ($year <= "1979")) {
    $dir = sprintf("3/item_%04d", $cnt3++);
}
if (($year >= "1980") && ($year <= "1989")) {
    $dir = sprintf("4/item_%04d", $cnt4++);
}
if (($year >= "1990") && ($year <= "1999")) {
    $dir = sprintf("5/item_%04d", $cnt5++);
}
if (($year >= "2000") && ($year <= "2100")) {
    $dir = sprintf("6/item_%04d", $cnt6++);
}

    # Make a directory for the item.
    print "mkdir $dir\n";
    system "mkdir $dir";

    # Create XML file from metadata
    open(fh,">:encoding(UTF-8)", "$dir/dublin_core.xml");
    print fh '<dublin_core>'."\n";
    print fh '<dcvalue element="identifier" qualifier="none">'
       .$key.'</dcvalue>'."\n";
    print fh '<dcvalue element="type" qualifier="none">Article</dcvalue>'."\n";
    print fh '<dcvalue element="language" qualifier="iso">en</dcvalue>'."\n";
    $affiliation = '';
    $affiliation1 = '';
    $affiliation2 = '';

    # Metadata for items with multiple authors, each
    # with individual affiliations, span multiple lines.
    # Collect them and produce XML for them.
    for $i (0 .. $#{$lt{$key}}) {
            @fields = split(/\t/, $lines[$lt{$key}[$i]]);
            $title = trim($fields[9]);
            if (length($title) > 0) {
                    $title =~ s/["]{1}([^"])/$1/g;
                    $title =~ s/("|"")$//g;
```

```
                print fh '<dcvalue element="title" qualifier="none">'
                    .$title.'</dcvalue>'."\n";
            }
        $year1 = trim($fields[1]);
            if (length($year1) > 0) {
                print fh '<dcvalue element="date" qualifier="issued">'
                    ."$year</dcvalue>\n";
            }
        $author = trim($fields[5]);
            if (length($author) > 0) {
                $author =~ s/(\$|\^|\{|\}|\*)//g;
                print fh '<dcvalue element="creator" qualifier="none">'
                    .$author.'</dcvalue>'."\n";
        }
            $abstract = trim($fields[10]);
            if (length($abstract) > 0) {
                print fh '<dcvalue element="description" qualifier="abstract">'
                    .$abstract.'</dcvalue>'."\n";
        }
            if (length(trim($fields[6])) > 0) {
                $affiliation1 = trim($fields[6]);
            }
            if (length(trim($fields[7])) > 0) {
                $affiliation2 = trim($fields[7]);
            }
            if ((length(trim($fields[6])) > 0)
                || (length(trim($fields[7])) > 0)) {
                if ((length(trim($fields[6])) == 0)
                    && (length($affiliation1) == 0)) {
                        $append = $affiliation2;
                } elsif ((length(trim($fields[7])) == 0)
                    && (length($affiliation2) == 0)) {
                        $append = $affiliation1;
                } else {
                        $append = $affiliation1.", "
                            .$affiliation2;
                }
                if (length($affiliation) > 0) {
                        $affiliation = $affiliation.
                            "; ".$append;
                } else {
                        $affiliation = $append;
                }
            }
            $note = trim($fields[11]);
            if (length($note) > 0) {
                print fh '<dcvalue element="description" qualifier="none">'
                    .$note.'</dcvalue>'."\n";
        }
    } # Done processing multiple authors.

    # Finish producing the XML for this item.
    print fh '<dcvalue element="description" qualifier="none">Author Institution: '
        .$affiliation.'</dcvalue>'."\n";
    print fh '</dublin_core>'."\n";
```

```
            close(fh);

            # Create the 'contents' file.
            open(fh, ">$dir/contents");

        if ($filename != "") {
                    print fh "$filename.jpg";
            $cmd = "cp \"../images/$year/$filename.jpg\" $dir";
                    print $cmd."\n";
                    system $cmd;
            }
        close(fh);
} # Finished processing this item.
```

-- import_collections.sh --

```
#!/bin/sh
#
# Import a collection from files generated on dspace
#
COLLECTION_ID=1811/6634
EPERSON="[name removed]@osu.edu"
SOURCE_DIR=./5
BASE_ID=`basename $COLLECTION_ID`
MAPFILE=./map.$BASE_ID

/dspace/bin/dsrun org.dspace.app.itemimport.ItemImport --add --eperson=$EPERSON --
collection=$COLLECTION_ID --source=$SOURCE_DIR --mapfile=$MAPFILE
```

Appendix C. Example DC XML for MSS 2009

<dublin_core>
<dcvalue element="identifier" qualifier="none">2009-MJ-10</dcvalue>
<dcvalue element="title" qualifier="none">VIBRATIONAL OVERTONE SPECTRA OF C_2H_6 AND C_2H_4 IN CRYOGENIC LIQUIDS</dcvalue>
<dcvalue element="date" qualifier="issued">2009</dcvalue>
<dcvalue element="description" qualifier="abstract">Vibrational overtone spectra of C_2H_6 and C_2H_4 in cryogenic solutions were recorded between 5000 and 14000 cm^{-1}. Spectral regions for the first four overtones were measured using a Fourier transform spectrophotometer. The fifth overtone $(\Delta\nu=6)$ spectra between 15,000 and 16,000 cm^{-1} were recorded with a double beam (pump-probe) thermal lens technique using concentrations as low as 10^{-3} mole fraction. The peak frequency shift $(\Delta\omega)$ from gas phase to solution is explained by the change in harmonic frequency and anharmonicity in solution with respect to the gas phase values. The bandwidth $(\Delta\omega_{1/2})$ of the $(\Delta\nu= 6)$ C-H absorption bands in solution can be explained in terms of collisions with the solvent molecules.</dcvalue>
<dcvalue element="description" qualifier="none">Author Institution: Department of Chemistry and Biochemistry, Baylor University, Waco, Texas, 76798</dcvalue>
<dcvalue element="type" qualifier="none">Article</dcvalue>
<dcvalue element="language" qualifier="iso">en</dcvalue>
<dcvalue element="creator" qualifier="none">Diez-y-Riega, Maria H.</dcvalue>
<dcvalue element="creator" qualifier="none">Manzanares, Carlos E.</dcvalue>
</dublin_core>

Final author manuscript of: Maureen P. Walsh, “Batch Loading Collections into DSpace: Using Perl Scripts for Automation and Quality Control,” *Information Technology and Libraries* 29, no. 3 (2010): 117-127.

Appendix D. Section of MSS Author Quality Control Script

-- flipper.pl --

```
#!/usr/bin/perl

#### Sections omitted ####

#### Begin author correction block ####

   $creatorxml = "";
   if (length($creators) > 0) {
         # Creator name are contaminated with comments.
         # Remove the comments.
     $creators =~ s/"//g;
         $creators =~ s/\\thanks\{.+\}//;
         $creators =~ s/\\thanks \{.+\}//;
         $creators =~ s/\\footnote\{.+\}//;
    # Multiple creators are separated by ';' or AND in the metadata.
    @creatorlist = split(/;| and | AND /,$creators);
         # Process each creator.
     foreach $creator (@creatorlist) {
           # Remove per name comments and punctuation.
       $creator =~ s/^\s+//;
           $creator =~ s/FULL NAME OF AUTHOR FROM OTHER LOCATION//;
           $creator =~ s/\\underline \{(.+)\}/$1/;
           $creator =~ s/\\address\{//;
           $creator =~ s/\\//g;
           $creator =~ s/\{//g;
           $creator =~ s/\}//g;
           $creator =~ s/\^//g;
           $creator =~ s/\'//g;
           $creator =~ s/\%//g;
           $creator =~ s/^AND$|^and$//;
       if (length($creator) > 0) {
                  $creator =~ s/\.(\w)/. $1/g;
                  # Split the name apart on spaces.
             @nameparts = split(/ /,$creator);
                  # Process each part of the name.
                  for($i = 0;$i <= $#nameparts; $i++) {
                     # Adjust case.
                     @nameparts[$i] = lc(@nameparts[$i]);
                     @nameparts[$i] = ucfirst(@nameparts[$i]);
                     $c = rindex(@nameparts[$i],"-");
                     # Uppercase hyphenated names.
                     if ($c != -1) {
                           $r = uc(substr(@nameparts[$i],$c+1,1));
                           substr(@nameparts[$i],$c+1,1,$r);
                     }
                  }
                  $lname = pop(@nameparts);
```

```
                $nl = @nameparts[-1];
                # Handle name prefixes.
                if ($nl eq "Von"
                   || $nl eq "Vander"
                   || $nl eq "Le"
                   || $nl eq "De"
                   || $nl eq "de") {
                   $lname = pop(@nameparts)." ".$lname;
                }
                # Handle special case name parts
                if ($nl eq "Der" ) {
                   $nl2 = @nameparts[-2];
                   $lname = pop(@nameparts)." ".$lname;
                   if ($nl2 eq "Van" ) {
                         $lname = pop(@nameparts)." ".$lname;
                   }
                }

                # assemble the name and make the XML.
                $name = $lname .", ".join(" ",@nameparts);
          $creatorxml .= '<dcvalue element="creator" qualifier="">'
             .$name.'</dcvalue>'."\n    ";
       }
    }
  } # Done processing creators of this item.


#### End author correction block ####

#### Sections omitted ####
```

Appendix E. MSS 2009 Batch Loading Scripts

-- mkxml2009.pl --

```
#!/usr/bin/perl

use Encode;                 # Routines for UTF encoding
use Text::xSV;              # Routines to process CSV files.
use File::Basename;

# Open and read the comma separated metadata file.
my $csv = new Text::xSV;
#$csv->set_sep('	'); # Use for tab separated files.
$csv->open_file("MSS2009.csv");
$csv->read_header();    # Process the CSV column headers.

# Constants for file and directory names.
$basedir = "/common/batch/input/mss/";
$indir = "$basedir/2009";
$xmldir= "./2009xml";
$imagesubdir= "processed_images";
$filename = "dublin_core.xml";

# Process each line of metadata, one line per item.
$linenum = 1;
while ($csv->get_row()) {
   # This divides the item's metadata into fields, each in its own variable.
   my (
         $identifier,
         $title,
         $creators,
         $description_abstract,
         $issuedate,
         $description,
         $description2,
         $abstract,
         $gif,
         $ppt,
   ) = $csv->extract(
         "Talk_id",
         "Title",
         "Creators",
         "Abstract",
         "IssueDate",
         "Description",
         "AuthorInstitution",
         "Image_file_name",
         "Talk_gifs_file",
         "Talk_ppt_file"
   );
```

```
$creatorxml = "";
# Multiple creators are separated by ';' in the metadata.
if (length($creators) > 0) {
        # Create XML for each creator.
    @creatorlist = split(/;/,$creators);
    foreach $creator (@creatorlist) {
        if (length($creator) > 0) {
            $creatorxml .= '<dcvalue element="creator" qualifier="none">'
            .$creator.'</dcvalue>'."\n    ";
        }
    }
} # Done processing creators for this item.

# Create the XML string for the Abstract.
$abstractxml = "";
if (length($description_abstract) > 0) {
        # Convert special metadata characters for use in xml/html.
    $description_abstract =~ s/\&/&amp;/g;
    $description_abstract =~ s/\>/&gt;/g;
    $description_abstract =~ s/\</&lt;/g;
        # Build the Abstract in XML.
    $abstractxml = '<dcvalue element="description" qualifier="abstract">'
        .$description_abstract.'</dcvalue>';
}

# Create the XML string for the Description.
$descriptionxml = "";
if (length($description) > 0) {
        # Convert special metadata characters for use in xml/html.
    $description=~ s/\&/&amp;/g;
    $description=~ s/\>/&gt;/g;
    $description=~ s/\</&lt;/g;
        # Build the Description in XML.
    $descriptionxml = '<dcvalue element="description" qualifier="none">'
        .$description.'</dcvalue>';
}

# Create the XML string for the Author Institution.
$description2xml = "";
if (length($description2) > 0) {
        # Convert special metadata characters for use in xml/html.
    $description2=~ s/\&/&amp;/g;
    $description2=~ s/\>/&gt;/g;
    $description2=~ s/\</&lt;/g;
        # Build the Author Institution XML.
    $description2xml = '<dcvalue element="description" qualifier="none">'
        .'Author Institution: ' .$description2.'</dcvalue>';
}

# Convert special characters in title.
$title=~ s/\&/&amp;/g;
$title=~ s/\>/&gt;/g;
$title=~ s/\</&lt;/g;

# Create XML File
```

```
    $subdir = $xmldir."/".$linenum;
    system "mkdir $basedir/$subdir";
    open(fh,">:encoding(UTF-8)", "$basedir/$subdir/$filename");
    print fh <<"XML";
<dublin_core>
    <dcvalue element="identifier" qualifier="none">$identifier</dcvalue>
    <dcvalue element="title" qualifier="none">$title</dcvalue>
    <dcvalue element="date" qualifier="issued">$issuedate</dcvalue>
    $abstractxml
    $descriptionxml
    $description2xml
    <dcvalue element="type" qualifier="none">Article</dcvalue>
    <dcvalue element="language" qualifier="iso">en</dcvalue>
    $creatorxml
</dublin_core>
XML
    close($fh);

# Create contents file and move files to the load set.

    # Copy item files into the load set.
    if (defined($abstract) && length($abstract) > 0) {
        system "cp $indir/$abstract $basedir/$subdir";
    }

    $sourcedir = substr($abstract, 0, 5);
    if (defined($ppt) && length($ppt) > 0 ) {
         system "cp $indir/$sourcedir/$sourcedir/*.* $basedir/$subdir/";
    }

    if (defined($gif) && length($gif) > 0 ) {
         system "cp $indir/$sourcedir/$imagesubdir/*.* $basedir/$subdir/";
    }

    # Make the 'contents' file and fill it with the file names.
    system "touch $basedir/$subdir/contents";

    if (defined($gif) && length($gif) > 0
        && -d "$indir/$sourcedir/$imagesubdir" ) {
        # Sort items in reverse order so they show up right in DSpace.
        # This is a hack that depends on how the DB returns items
        # in unsorted (physical) order. There are better ways to do this.
        system "cd $indir/$sourcedir/$imagesubdir/;"
            . " ls *[0-9][0-9].* | sort -r >> $basedir/$subdir/contents";
        system "cd $indir/$sourcedir/$imagesubdir/;"
            . " ls *[a-zA-Z][0-9].* | sort -r  >> $basedir/$subdir/contents";
    }

    if (defined($ppt) && length($ppt) > 0
        && -d "$indir/$sourcedir/$sourcedir" ) {
        system "cd $indir/$sourcedir/$sourcedir/;"
            . " ls *.* >> $basedir/$subdir/contents";
    }

    # Put the Abstract in last, so it displays first.
```

```
    system "cd $basedir/$subdir; basename $abstract >>"
        . " $basedir/$subdir/contents";

    $linenum++;

} # Done processing an item.
```

-- import.sh --

```
#!/bin/sh
#
# Import a collection from files generated on dspace
#
COLLECTION_ID=1811/6635
EPERSON=[name removed]@osu.edu
SOURCE_DIR=./2009xml
BASE_ID=`basename $COLLECTION_ID`
MAPFILE=./map-dspace03-mss2009.$BASE_ID

/dspace/bin/dsrun org.dspace.app.itemimport.ItemImport --add --eperson=$EPERSON --
collection=$COLLECTION_ID --source=$SOURCE_DIR --mapfile=$MAPFILE
```