|  |
| --- |
| GEOG-596B Project Report |
| Python-based Solutions to Maintain Enterprise Data Currency at the Bureau of Land Management |
| Adam Ridley, GIS Specialist |

## Introduction

For my capstone project, I chose to focus on developing Python scripts to enhance GIS data management processes within the Idaho Bureau of Land Management (BLM). Idaho BLM has a unique GIS system architecture driven by historic issues with Wide-Area Network (WAN) bandwidth in a number of our field offices. While other BLM states have been reasonably successful in conversion to Citrix-based centralized GIS services and data, infrastructural bandwidth restrictions have kept centralized services from being an effective option for the Idaho BLM. Instead, we use an Esri-based, tiered, one-way Spatial Database Engine (SDE) replication process whereby the thirteen Field Offices throughout Idaho receive periodic replicas of changes to the corporate data held at the State Office. Alternately, Field Offices provide updates and edits to corporate data through whichever means their infrastructure supports, typically either direct edits to SDE versions or local database Check-Outs/Check-Ins from SDE versions. Regardless, a common challenge for field offices throughout the state is keeping each library of ancillary files (layer files, metadata, map documents) that reference our corporate data up to date as changes are implemented. That is where I believe my capstone work will be able to make a difference.

## Project Background

### Data Management Challenges

In discussions with the Idaho BLM GIS Lead, I identified four distinct data management challenges to be addressed through this project:

* **Challenge:** Layer files reference incorrect data (out of date, wrong location, etc.) or have broken links due to schema changes.

**Solution:** Validate corporate layer files and fix or remove broken links.

* **Challenge:** Despite scheduled nightly replication processes in place, local corporate datasets become decadent due to problems with replication.

**Solution:** Test our local corporate geodatabases against the State Office data stack for currency and update as needed.

* **Challenge:** Metadata is not typically included in the one-way replication process used to maintain data currency and datasets are too large for frequent, wholesale replacement.

**Solution:** Retrieve current metadata from the State Office data stack to replace local corporate data as updates are completed at the State Office.

* **Challenge:** Changes to local corporate data break links or symbology in pre-existing map documents (MXD files) referencing affected datasets.

**Solution:** Test MXD files for broken links and repair or replace data as needed.

### Layer File Validation

Layer files are a file format created by Esri which provide easy linking to datasets with preset symbology and layer properties. The Idaho BLM utilizes layer files to simplify the process of connecting end-users with the data they need. We maintain a library of layer files on each local server that point to our replicated, state-wide datasets and any non-standard, local reference datasets, which are collectively referred to as “Final Data.” Many end users consume data through our layer file repositories either out of convenience, due to lack of familiarity with our data stores, or due to limited experience with ArcGIS. As a result, maintaining layer file currency and function is an important business need. Historically, local GIS Specialists have been tasked with developing and maintaining these layer file libraries, but, with the increasing standardization of our data at the state and national levels, a growing number of layer files are created by GIS staff within the State Office or at the National Operations Center (NOC).

Layer files work by storing a snapshot of a dataset’s properties, such as file path, symbology, etc., as seen in an MXD file. When one of the properties of the underlying dataset changes layer files can become inoperable and require re-linking. While our Final Data stores are relatively stable, changes to its structure and contents are not uncommon and are not always communicated. Therefore, checking that layer files have valid file paths and symbology linkages becomes an important and potentially time-consuming endeavor.

### Replica Currency

Replication is a process created by Esri to manage changes to spatial data across an enterprise GIS designed to provision data for many users within an organization. Idaho BLM uses replication to push data from the State Office GIS server to servers located at Field Offices throughout the state. Currently, there is a Python-scripted process in place to push changes from the State Office geodatabase to the Field Office replicas nightly.

However, there are a number of issues with this model. At present, not all datasets contained in the State Office geodatabase are set to replicate regularly. This can lead to an inappropriate perception on the part of GIS Specialists and end users that all datasets are current, despite the possibility that they may not have been replicated for several months. Additionally, the nightly replication process can fail for a variety of reasons, often due to active schema locks on datasets from a connected user. Regardless of the root cause, these replication failures are largely unseen and unreported at the Field Office level. Typically, a majority of the replicated data is static, experiencing either infrequent, significant changes or regular, minor changes. While generally a low-priority issue, data currency becomes critical for GIS workflows pertaining to emergency management and National Environmental Policy Act (NEPA) planning. To that end, Field Office GIS Specialists need to be aware of our local replica’s currency and have a mechanism for updating datasets that have lapsed.

### Metadata Currency

Similar to the issues with replica currency, metadata associated with our Final Data stores can often be an unknown quantity. Metadata stability within our Final Data tends to mirror the stability of the spatial and tabular components of each dataset. Therefore, some replicated datasets will have relatively unchanging metadata while others may see frequent edits. Unfortunately, the process by which spatial and tabular changes are communicated from the State Office to local servers, referred to as “replica synchronization,” does not encompass changes to metadata. As a result, updates to Field Office replica metadata occur only when the replica is rebuilt or when changes are pushed manually by a GIS Specialist. Either scenario is executed on an infrequent-at-best time scale, creating uncertainty regarding the currency and accuracy of metadata for replicated datasets.

### MXD Data Validation

Much of the work accomplished by GIS end-users at Idaho BLM is arranged around individual NEPA projects, which are often multi-disciplinary in nature, but focused around a specific management need, such as timber sales, weed treatments, realty leases, etc. To ease navigation and allow for simple archiving, project-related GIS files are organized by individual NEPA project (Figure 1) with standard subfolders (Figure 2).



Figure 1- GIS Directory Structure



Figure 2 – Standard SubFolders

GIS analysis for these projects typically uses a combination of both reference data from Final Data and project-specific data, which are stored in separate folder structures on our local servers. Data referenced in MXD files store file paths one of two ways: as an absolute path, which stores the full directory paths of data, or as a relative path, which stores the paths of data relative to the location of the MXD file. Regardless of which system we choose, links to either reference data or project data will be broken if files are moved or renamed. This presents a continual issue in considering the long-term viability of ArcMap documents and reliable access for archived map documents. End-users frequently struggle to decipher where to re-link broken data paths and often require help from GIS Specialists to reconstitute old MXDs, with varying levels of success depending upon age and naming conventions.

## Literature Review

### Reviewed Literature

There are a limited number of publications or papers related to the use of Python for GIS data management. In conducting a literature search, I found that most of the pertinent work I was able to identify and retrieve exists either as presentations from paper sessions and technical workshops at Esri conferences or as white papers included in Esri publications. Ideally, I would like to have located a wider range of articles, which included sources outside of Esri-sponsored communications. However, given this project’s integration with the ArcGIS software ecosystem, it may not be possible or necessary to find pertinent literature outside of the aforementioned sources.

While I found a limited number of publications which were directly relevant to my capstone project, a few paper sessions were helpful in implementing my project. Watkins’s (2014) work detailed one approach to traversing and interrogating data stored in SDE-based geodatabases using the *arcpy.da.Walk* function and ArcPy Describe objects. I ended up utilizing a similar approach to investigate data in each of the main scripts I created. In addition, Hickey (2009) discussed a problem with similar bounds to those I’ve identified in terms of maintaining data currency across parallel, dislocated servers, but with a focus on city addresses and county parcel data.

### Additional Sources

Beyond the somewhat more formalized sources discussed above, I made extensive use of a number of threaded discussion forums, the built-in ArcGIS Help documentation, and the Python Software Foundation documentation. GIS Stack Exchange, Stack Overflow, and Esri’s GeoNet forums proved to be indispensable throughout the script development process. In particular, I adapted the idea of creating an empty XML file and importing metadata contents into the file from a Stack Exchange post (blah238, 2013).

## Implementation Approach

While each challenge could be addressed by the development of a standardized, manual workflow repeated by staff at each field office, the use of scripting to accomplish similar results provides a number of advantages. Scripts allow for the processes to be executed outside of regular business hours thereby reducing the likelihood for schema locks and competition for network resources, which can be particularly limited during regular business hours. In addition, scripting abstracts some amount of input that would be required from a GIS Specialist in the manual execution of each task, which consequently reduces personnel time and costs. Finally, due to the need for routine execution of the tasks, scripting allows for increased consistency in terms of frequency of task execution and content.

There are a number of potential avenues for automating GIS data management tasks including ArcGIS Add-Ins, stand-alone applications, geoprocessing models created in ArcGIS’s ModelBuilder, and Python scripts. Department of Interior (DOI) and Bureau of Land Management IT policy requires applications pass a complex vetting process called Configuration Management (CM) before they are approved for use on agency hardware and networks. The CM process is often lengthy and backlogged, resulting in potential multi-year delays in implementation.

For those of us bound by DOI and BLM IT policy, Python scripting has the advantage of not relying on dynamic link library (.DLL) or executable (.EXE) files, both of which would necessitate CM vetting. Furthermore, ArcGIS has increasingly robust integration with Python through the ArcPy site package and subsidiary modules. ArcGIS ships with the option to install Python and, with it, the IDLE integrated development environment, which effectively provides an environment for all BLM GIS Specialists to develop and modify Python scripts without additional expenditures in software.

In order to maximize the utility of the code produced through this project and develop a product that could be used by all Idaho BLM GIS Specialists, my goal was to make the overall product as modular and user-friendly as possible. To that end, I created each script such that it could be executed from IDLE, from Windows command line, or as ArcGIS Script Tool. Additionally, I worked to provide users with the ability to identify issues in their data, but resolve them manually, if they were uncomfortable with the prospect of the script fixing issues without user guidance.

Finally, in designing each script, I was careful to use only modules provided as part of the standard Idaho BLM ArcGIS and Python installations. This runs somewhat counter to my initial plan to use the wxPython site package to build a graphical user interface (GUI) for the tools. In the end, I felt it was better to design the scripts to run from our standard installation and not require additional site packages than to build a stand-alone GUI for the scripts.

## Script Specifics

### Layer File Validation

The process for validating layer files requires three main components: a reliable way of indexing through our layer file repository, a method for checking whether each layer file has a valid file path and symbology reference, and a method to locate the correct data source should the link be broken. For the first component, I employed the *arcpy.da.Walk* function to selectively search for layer files within our Final Data directory. The *ListBrokenDataSources* function under *arcpy.mapping* then provides the ability to determine whether any given layer file is active and valid. The layer file validation script then employs a path-based search methodology, which it shares with the MXD validation script, called FixLinks. FixLinks uses information from the original, broken data source to help determine where to search for the proper dataset to re-link the file. If the original data source path contains references to either of our two main reference data stores, Final Data or SDE Master Data, it directs its search to those locations. Otherwise, FixLinks assumes the layer accesses project data and searches based on the original data source path. FixLinks starts searching at the original data source location, but recursively moves to the parent directory if no match is found. Searching continues until a match is found, a preset directory file structure location is reached, or the user interrupts the process. Regardless of the location, the layer file validation script employs FixLinks to consider only exact filename matches to the original data source and is therefore not suited to addressing link breakage resulting from changed filenames.



Figure 3 – Layer File Validation Script Tool Interface

The script divides broken layer files it finds into three categories: Fixable, Unmatched, and Trouble. Fixable files are items for which the script was able to find an exact match in our reference data stores. Unmatched files are items for which the script could not find a match. Trouble files are layer files which presented some difficulty to the script. Most commonly that means the file doesn’t contain a data source to query or the data source pointed to an inaccessible location. Upon completion, the script generates a text file report which lists the broken layer files by category (see Appendix B). Finally, users have the option to automatically repair the paths for any layer files which end up in the Fixable category (see Figure 3).

### MXD Data Validation

The process for MXD validation largely mirrors the process developed for layer file validation. The *arcpy.da .Walk* function is employed to locate and iterate through MXDs within the input workspace. *ListBrokenDataSources* (*arcpy.mapping*) is used to populate a collection of *Layer* objects which have broken links. The script then uses the aforementioned FixLinks module to search for the appropriate data source to re-link each broken layer. Upon completion, the script generates a text file report which lists the broken layer files by category.



Figure 4 – MXD Validation Script Tool Interface

The MXD Validation script accepts three inputs: one required and two optional (see Figure 4). Users can provide the script with either a single MXD file or a folder. If given a folder, the script will search the provided location and any sub-folders therein for any MXD files it can locate and tests them one by one. Otherwise, the script checks the single MXD file provided. The script creates a report text file, similar to that generated by the layer file validation script, in the input workspace provided by the user (see Appendix B). For each MXD file, layers with broken links are divided into Fixable, Unmatched, and Trouble categories. Users have the option to automatically repair the paths for any layers which end up in the Fixable category. The MXD file validation script offers an additional option, called Quiet Mode. Quiet mode was developed to allow users to execute the validation process with minimal input, if desired, or allow for additional direction from the user to enhance the data matching process. While running with Quiet mode “on,” the script will consider only exact matches to the name of the original feature class in order to minimize the potential for incorrect matches. While running with Quiet mode “off,” FixLinks will consider features classes with names that are 50% similar to the original feature class, as well as prompt users to review and verify any potential matches. The fuzzy matching mechanism is provided by the *get\_close\_matches* function built-in to the *difflib* Python module, which is part of the standard Python library. The *get\_close\_matches* function performs comparisons between a reference string and a list of test strings, with options to set a matching threshold and a limit on the number of results returned (Python Software Foundation, 2016). For our purposes, FixLinks considers only dataset names that are a 50% or greater match to the original feature class

While the MXD Validation script and Layer File Validation script both use FixLinks and could access Quiet mode, I made the decision to hard code Quiet mode “on” for the Layer File Validation script tool. I did so due to the likelihood of a user having to wade through a large number of close matches from similarly named, but distinct datasets. As an example, using FixLinks to search for WLDLFE\_SageGrouseHabitat\_PUB\_100K\_POLY\_2007 with Quiet mode “off” would create matches with 11 other datasets with similar names, but different years. Or datasets with multiple geometries, such as VEG\_OTH\_IFWISPlantPCD\_INT\_24K\_LINE and VEG\_OTH\_IFWISPlantPCD\_INT\_24K\_POLY, may return false matches. Either way, Quiet mode requires the user to respond to each potential match until a match is chosen or all the potential matches have been rejected. While that may be desirable for validating MXD files, it seemed overly burdensome for validating layer files. However, Quiet mode could be easily implemented in the Layer File Validation script, if so desired.

### Replica Currency

Testing the currency of our local SDE replicas against the parent geodatabase at the State Office requires two main components: a means by which to index through both the replica and parent geodatabases, and a means to test analogous feature classes’ similarity. I used the previously mentioned *arcpy.da.Walk* function to iterate through the feature datasets, feature classes, and tables within each geodatabase.



Figure 5 – Replica Currency Script Tool Interface

In considering methods for comparing datasets between geodatabases, I tried to present the user with two options which provide differing levels of scrutiny in determining whether a dataset has changed and requires updating. At the simplest level of scrutiny, the script compares the number of features between geodatabases and considers the database with a greater number of features to be more current. There are some obvious cases where the logic of using feature counts breaks down, such as: feature aggregation, deletion of inaccurate or non-existing features, etc. Despite those potentially challenging scenarios, feature counts still provide a useful baseline for change detection, which would alert a Field Office GIS Specialist to a discrepancy between databases. I am continuing to evaluate other metrics for a simple comparison, including using the Modified Date property of the feature class.

At a higher level of scrutiny, the built-in Feature Compare tool (Data Management > Data Comparison > Feature Compare) provides functionality to compare features based on their geometry, attributes, schemata, spatial references, or all of the aforementioned properties (Esri, 2016). One of the advantages of this tool is that it executes a tiered series of tests, looking first at spatial reference, then geometry, then schema, and finally attributes. However, there are some notable limitations to the suitability of Feature Compare in the Idaho BLM environment. For our convenience in the field offices, the main publication database at the State Office, IDP1V, is projected from Idaho Transverse Mercator to the local UTM zone of the field office during replication. Unfortunately, Feature Compare is not capable of contrasting geometry between datasets with different spatial references. As a result, the Detailed Compare option built into the replica currency script only performs table comparisons. Testing revealed that executing repetitive table comparisons across the WAN made for a lengthy process, requiring over 20 hours to complete and utilizing significant network resources. Therefore, unless I can devise a less network-intensive approach, the utility of the Detailed Compare option is limited.

Some of the work considered through my literature review pointed to the use of the *arcpy.Describe* function to access properties related to any given data element (Watkins 2014). However, for the purposes of comparing feature currency, it does not appear to provide access to some of the more useful metrics, such as feature counts, and may not be of much use for this application.

Similar to the other scripts, the replica currency script produces a report text file containing results (see Appendix B). When running a simple comparison, the script reports which feature classes contain equal counts between the two servers, which datasets are more current locally, which datasets are more current in SDE, and if there are any datasets in SDE which do not have a local equivalent. If the user has chosen the “Replace Local data” option, datasets which are more current in SDE are copied to the local replica. Alternatively, until a less network-intensive process can be worked out, the results from a Detailed Compare print only to the console or ArcGIS geoprocessing window.

### Metadata Currency

The overall process for testing metadata currency is conceptually similar to testing the currency of the data underlying the metadata. The first step in the process is to develop a mechanism to iterate through our local replica and IDP1V to obtain metadata for each dataset in each database. At the start, the metadata currency script creates a temporary connection file to IDP1V and two separate metadata XML file repositories for local metadata and metadata from IDP1V. Once again, I employed the *arcpy.da.Walk* function to access the datasets with our local replica and IDP1V. As the *arcpy.da.Walk* function indexes through each dataset, the script creates a XML file named after each dataset into which the feature class’s current metadata is copied using Esri’s Metadata Importer tool. This process allows for a direct copy of each feature class’s metadata to be made without requiring the use of a translator or multiple-step conversion and reconversion process that would result from using the Export Metadata tools.

The second step in the process is to compare equivalent metadata files to determine which is more current. This is accomplished by parsing the XML element tree and searching for the ModDate element for each XML file. The script then matches datasets by name between databases and compares each file’s ModDate to assess currency.

Finally, the script creates a report text file that denotes which datasets have current metadata in the local replica and which are more current in IDP1V (see Appendix B). If the user has chosen the “Replace Local” option, any datasets that are more current in IDP1V will have their metadata replaced in the local replica.



Figure 6 – Metadata Currency Script Tool Interface

There are two notable limitations to the operation of this script. The first issue is that it can only be executed from a 32-bit Python environment. The Esri-built metadata tool it uses (Metadata Importer) will not function from a 64-bit environment. At present, the script has not been designed to test for its execution environment, 32-bit or 64-bit, and, until there is a programmatic solution, user education will be employed to avoid possible issues. The second issue of note is the limitation imposed by using the ModDate element of the metadata as a currency metric. By default, any time a user views metadata for an item in ArcCatalog, the ModDate is updated. However, this feature can be turned off from the ArcCatalog Options > Metadata tab. The usefulness of this script will be greatly enhanced by users turning that feature off and, in the absence of a global fix, users will be encouraged to do so.

## Post-Implementation

With development and implementation complete, the next issues to address are: How do these tools get into the hands of their intended users, and what kind of support will be provided for these tools going forward?

### Distribution

The distribution approach will be to make the scripts available to one field office at a time. At present, the scripts have been run extensively on the Cottonwood Field Office server and in limited tests on the Shoshone Field Office server. Ideally, I would like to have more Field Offices test the function of each script before making the tools available to the rest of the Idaho BLM. It is my hope that, through the limited release process, some additional bugs and issues with the scripts can be discovered and addressed before a wider distribution. In doing so, I hope to avoid the need to localize any given script for a particular field office, thus allowing for easier distribution of updates in the future.

To aid in the distribution process, I have placed the scripts in a repository on GitHub (see Appendix A) and will use GitHub’s built-in functionality to manage and distribute changes going forward. GitHub also offers the added benefit of allowing GIS Specialists in other offices to contribute to the suite of scripts through forking and modification.

### Support

In the months following implementation and during the limited-release process, a store of support documentation will be developed for the scripts. The documentation will include script details and workflows with the intended audience being Field Office GIS Specialists seeking to use the tools. The scripts have been internally documented through commenting, which will be updated and adjusted as revisions to the scripts are completed.

Beyond documentation, I anticipate that some training may be necessary, especially with those specialists participating in the initial, limited release. That training will likely be provided through phone conversations, instant messaging, and video conference due to the distances involved and cost of travel and will be conducted on an individual basis as the scripts are distributed.

### Updates

As indicated throughout the implementation section, I have already identified a number of ways in which the scripts can be improved and I expect the limited test phase will illustrate additional needs. Once testing and agency-wide rollout is complete, I will consider the potential improvements I’ve identified to that point and request additional input from other Field Offices for needs they have observed. If time allows, updates would ideally be completed after field season, in the November-February timeframe. It can be hard to gain administrative support for coding projects, but use of the scripts throughout Idaho BLM would be the best argument for supporting these endeavors.

At present, the scripts are compatible with ArcGIS 10.2.2 and ArcGIS 10.3.1. I suspect they should run without issue in ArcGIS Pro as well, but compatibility will need to be confirmed as ArcGIS Pro becomes available for Idaho BLM use. Looking to the future, it is difficult to predict whether the scripts will require updates to function in upcoming versions of ArcGIS and whether there will be administrative support to revisit the code at that time.

## Summary

This project presented a variety of challenges and learning opportunities throughout the development and implementation phases. From a coding standpoint, I was able to expand my knowledge of ArcPy and Python, while further refining my approach to coding projects. Specifically, this project gave me a greater understanding and level of comfort with the fundamentals of Python in terms of string manipulation, a variety of looping techniques, efficient coding through defining functions, and file output. Furthermore, this experience required me to develop code for a larger audience than my previous work, which necessitated anticipating users’ needs and building some input validation into the scripts.

Beyond coding, there were a number of “take-aways” in terms of project management. I feel like this project would have benefited from additional input or direction from the agency and the wider Idaho BLM GIS community. Unfortunately, due to turnover at the State Office and the agency’s current focus on other management issues, such as Greater Sage Grouse concerns, the people best suited to advising this effort were otherwise committed. Given the opportunity to start over, I would seek to establish more consistent communication with an appropriate agency advisor regarding the content and direction of the project from the outset.

Overall, this was an interesting and fruitful capstone project which I hope will provide tangible benefits to the Idaho BLM. Furthermore, it was a challenging project to undertake, given the minimal support available from the agency and the relatively short development timeframe. I have received a number of queries regarding the availability of my scripts since the presentation, which is encouraging. I look forward to working with those interested parties in helping them make the best use of these tools and I plan to garner input to make the next version even better, given support to do so. Ultimately, I believe these scripts serve to give myself and other Field Office GIS Specialists a better understanding of our data and are a significant step toward addressing the data management challenges we face.

## References

blah238. “Creating table containing all filenames (and possibly metadata) in File Geodatabase?” GIS Stack Exchange. Web. 4 May 2016.

Esri. “ArcGIS Help 10.2, 10.2.1, and 10.2.2.” Esri. Web. 4 May 2016.

Hickey, Michael. Keeping Users' Data Current Using Geoprocessing Models & Python Scripts. 2009 Esri User Conference Proceedings.

Python Software Foundation. “Python 2.7.11 documentation.” Python Software Foundation. Web. 4 May 2016.

Watkins, David. Using Python to Gather Information about Data in SDE. 2014 Esri User Conference Proceedings.

## Appendices

### Appendix A – Source Code

The source code for this project is currently available on GitHub at: <https://github.com/ajridley/RidleyTools.git>

### Appendix B – Script Report Examples

#### Layer File Validation



#### MXD Validation



#### Replica Currency



#### Metadata Currency

