2016

GIS Data Preparation & Integration: Scripts & Functions Documentation

COLORADO WEST REGION DATA INTEGRATION GROUP

HEATHER WIDLUND, GIS COORDINATOR, SAN MIGUEL COUNTY, CO

HEATHERW@SANMIGUELCOUNTYCO.GOV

HTTPS://GITHUB.COM/HWIDLUND/INTEGRATION

CONTENTS

PROJECT DESCRIPTION

BACKGROUND

Disasters and emergencies don't stop at administrative boundaries. However, important local Geographic Information Systems data are generally maintained only within each agency's area of responsibility, such as a city or county. While it is increasingly recognized that broader sets of these GIS data are critical to emergency situations, there is a lot of variation in how (or if) local agencies deploy and share data, much less integrate them.

COLORADO WEST REGION INTEGRATION GROUP

The Colorado West Region Data Integration Group (see Figure 1) succeeded in developing unified data schemas for framework layers with the specific goal of supplying cross-jurisdictional data for emergency response. The integrated data are now in active use for visual mapping in 9-1-1 dispatch centers, verifying an emergency notification address database, maintaining 9-1-1 database records, and are supplied to the Colorado Office of Information Technology (OIT) for statewide data aggregation.

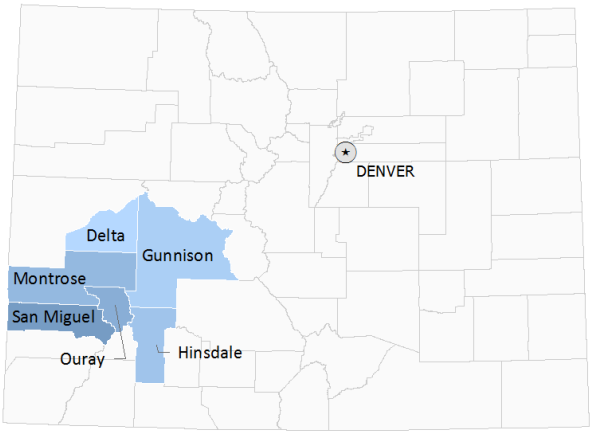


Figure 1 Agencies participating in Colorado West Region Data Integration Group partnership (2016)

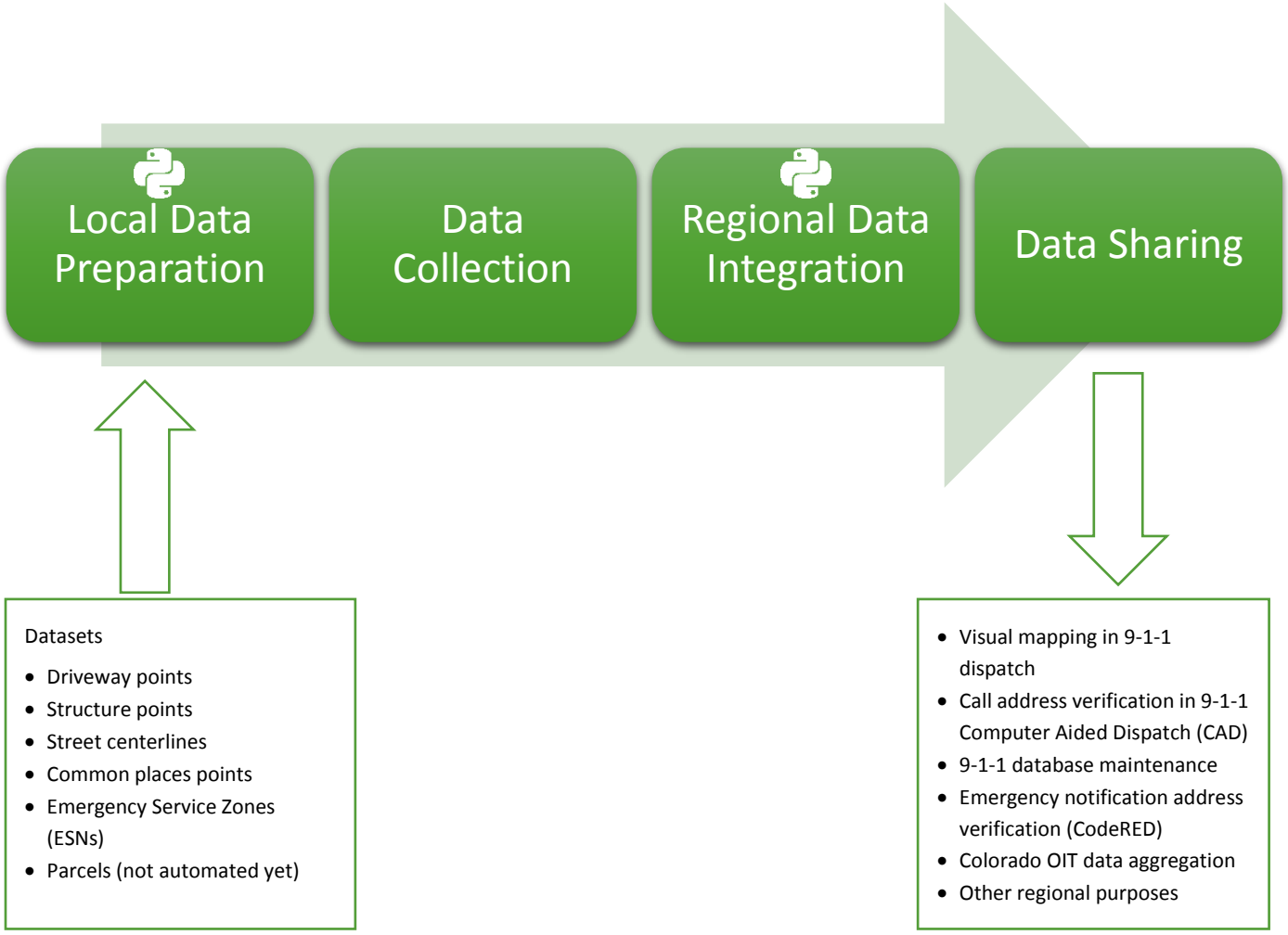
Still, data must be preprocessed at each source agency to comply with the schema, which has resulted in infrequent data updates. Infrequent data updates could lead to dispatching errors and incomplete emergency notifications. This project addresses the issue by automating data preparation and integration.

FIELD NAME	NULLABLE	FIELD SOURCE STANDARD	MAX CHAR	TYPE	ALIAS	DESCRIPTION
SOD	NO	i3	6	Text	Source of Data	Agency who last updated the data, usually the 911 authority
SID	NO	i3/NENA	20	Text	Unique site ID	e.g. SANMCO_XXXXX
EDT	YES	NENA	N/A	Date	Effective Date	Date address is effective
SAN	YES	NENA/PIDF-LO	(10)	Long Integer	Site Address Number	Numeric identifier of a location along a road
SNS	YES	PIDF-LO	5	Text	Site Address Number suffix	A, 1/2, etc.
PRD	YES	roads data dict./NENA	2	Text	Street Prefix Direction	(n,e,s,w nw,ne,sw,se)
RD	NO	roads DD/NENA	60	Text	Street Name	spelled out street name
STS	YES	roads DD/NENA	4	Text	Street type suffix	from postal standards (e.g. ST, AVE, RD, WAY)
POD	YES	roads DD/NENA	2	Text	Street Post Directional	(n,e,s,w nw,ne,sw,se)
FSA	NO	group	100	Text	Full Street Address	Full street address with all elements incl. unit
ESN	NO	NENA	(3)	Short Integer	Emergency Service Number	Number identifying the combination of Law, Fire, EMS services for the location.
MCN	NO	NENA	35	Text	MSAG Community Name	Community name used on the address's related MSAG Range and associated ALI data. For geocoding of ALI data. May not match Postal Community Name.
PCN	NO	i3	35	Text	Postal Community Name	Community name used for USPS delivery or ZIP.
BLD	YES	i3	35	Text	Building Name	Common name of building/location
UNIT	YES	i3	5	Text	Unit ID	Unit identifier, apart from Address Number Suffix
UTY	YES	i3	35	Text	Unit type	Unit, Apt, Trlr, Suite
LOC	YES	i3	35	Text	Location Information	Additional location info/comments
PLC	YES	PIDF-LO	35	Text	Place Type	(office, school, store, church, residential, mixed)
LAT	YES	group	(15)	double	Latitude, GCS Decimal Degrees	Double numbers allow up to 15 places. Minimum
LON	YES	group	(15)	double	Longitude, GCS Decimal Degrees	six place past the decimal.

Figure 2 Address & structure point data schema (partial)

INTEGRATION TOOLS

A set of automated tools has been developed using Python, the arcpy library, and ArcGIS script tools, with the goal of increasing the frequency and completeness of data updates. Automation has been applied to two stages of the process: data preparation at the source agencies, and integration of the prepared data. The data are still collected manually after preparation (currently using a shared Google Drive location), and distributed manually after integration.



TERMS & ABBREVIATIONS

Function	Block of code within a script to execute a specific task(s). Reduces repeated code.
Script	File with a *.py extension which is executable by Python. Can contain functions or simple code.
SID	Unique Id: may refer to field name or object
SOD	Source of data: may refer to field name or object

ORGANIZATION & SETUP

There are two sets of Python scripts and tools: one for **source data preparation** and one for **regional data integration**.

The functions in the preparation scripts are designed to extract applicable data from each data source (agency) and transform into datasets which match the agreed-upon data schemas. The outcome of the preparation stage is a zipped file geodatabase containing all data layers in the integration schema for a single agency.

The functions in the integration scripts are designed to process the output from each agency's preparation (a zipped file geodatabase) and combine the data into seamless datasets. The outcome of the integration stage is a file geodatabase containing all data layers in the integration schema for all agencies in the group.

DEPLOYMENT PACKAGE

A directory structure is deployed in each agency, consisting of four directories, an ArcGIS Toolbox, and a batch file. This location should be permanent and accessible by the machine which will execute the scripts.

The *config* folder will contain configuration files (created by the ArcGIS Integration Tools toolbox); the log files will be housed in the *logs* folder; the provided blueprint geodatabase for the source agency is located in the *outputdata* folder, and the scripts are in the *scripts* folder.

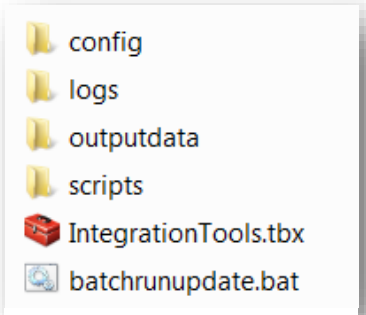


Figure 3 Deployment package

REQUIREMENTS

- A computer to execute scripts which has Python 2.x and ArcGIS 10.2+ installed
- Deployment package provided by author and housed in a permanent and accessible location
- Source data
- ArcGIS applications should be closed during processing

PREPARATION SCRIPTS & FUNCTIONS HIERARCHY DIAGRAM

(scriptname.FunctionName)

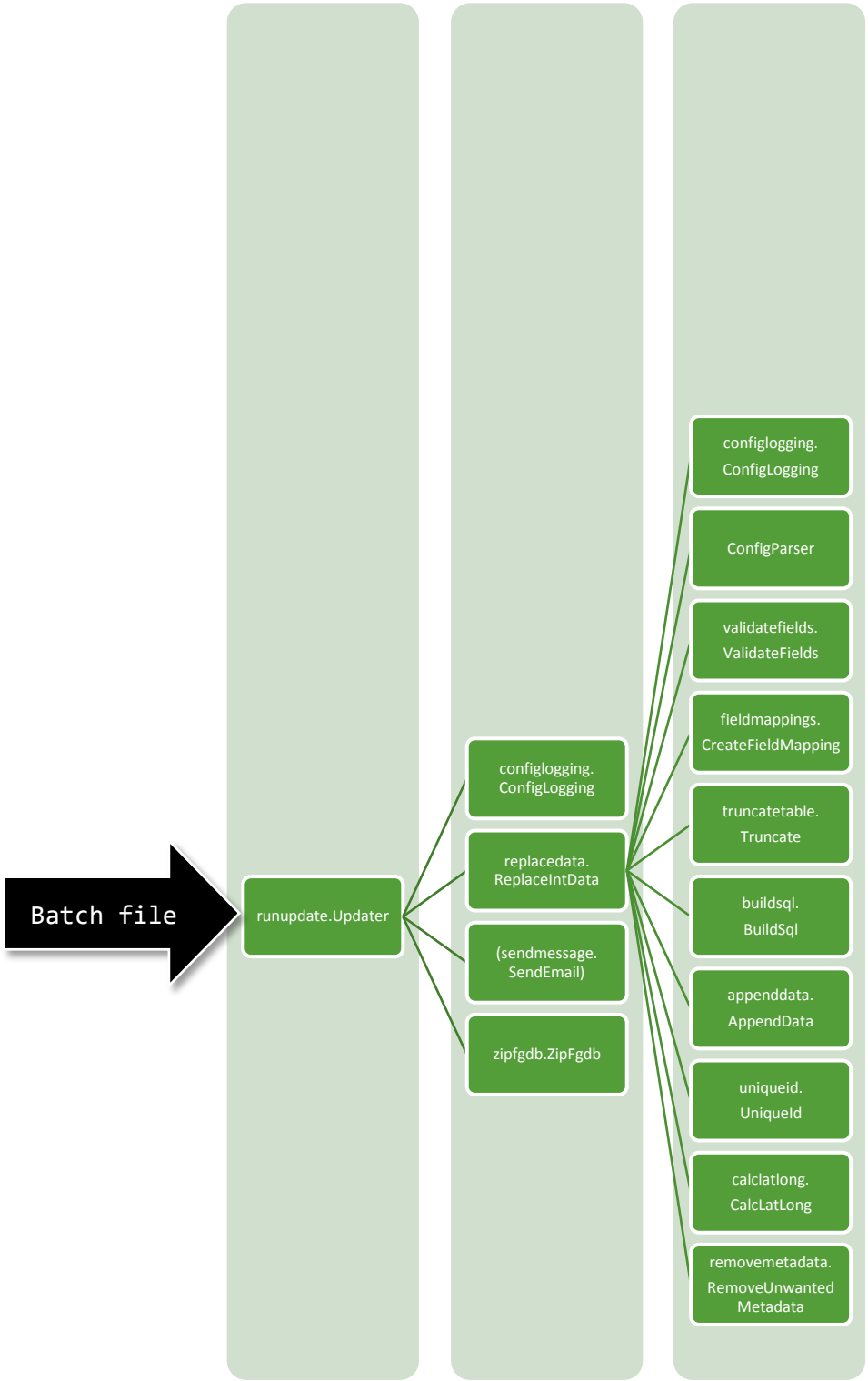


Figure 4 Preparation Scripts & Functions Hierarchy

FUNCTIONS

Note: Function inputs are named per the names of the variables in the scripts: the names of inputs vary between calling and executing functions. In other words, what is listed in parentheses might not match what is listed in "inputs".

Updater

FUNCTION

Updater(configPath, logFilePath, dataPath)

INPUTS (PASSED TO FUNCTION BY BATCH FILE)

configPath	Full path to directory where *.ini files to be processed are located
logFilePath	Full path to directory where log files will be located; will be created if doesn't exist
dataPath	Full path to directory where output data geodatabase is located

DESCRIPTION

Triggered by batch file. Iterates through all *.ini files in a directory, calling the **ReplaceIntData** function on each to process its associated feature class.

If the **ReplaceIntData** function results in an error, a True/False variable is toggled to True. Otherwise, the function zips the newly processed geodatabase in the output data directory. If an error condition exists (i.e. sendError=True), the other *.ini files/feature classes will be processed, but the geodatabase won't be zipped and the batch script window will stay open and show a message to check log files for the problem.

IMPORTS (OTHER FUNCTIONS THAT ARE CALLED TO DO ADDITIONAL PROCESSING)

ConfigLogging	Configures the log files' setup for detailed activity logging
ReplaceIntData	Performs the actual data cleanup and replacement
SendEmail	Optional function for notification emails (requires SMTP server credentials)
ZipFgdb	Zips the geodatabase after the successful outcome of all the **ReplaceIntData** loops

CONTAINING SCRIPT

runupdate.py

ConfigLogging

FUNCTION

ConfigLogging(logFileLocation)

INPUTS

logFileLocation	Full path to directory where log files are located

DESCRIPTION

Creates a Python dictionary to configure logging, with logging settings for different levels (which vary in verbosity). This function allows the calling or "parent" function to create a logging "object" enabling the calling function to write messages to files. The **Updater** & **ReplaceIntData** functions can log to a file. The "child" functions pass messages to those two to write to log files. The log files will be created if they don't exist, and appended to otherwise. To "clean the slate", simply delete the old log files.

The two levels of log are "debug" and "warn", which are written to debuglog.log and warnlog.log in the logs directory and can be opened in Notepad. The debug level reports detailed progress messages in addition to errors, while the warn level contains just warnings and errors. Warnings will allow the functions to continue to run; for example, duplicate values in the SID field, while errors will stop execution.

CONTAINING SCRIPT

configlogging.py

ReplaceIntData

FUNCTION

ReplaceIntData(config_file, log_file_location)

INPUTS

Config_file	Full path/filename of current *.ini file
Log_file_location	Full path to directory where log files will be located; will be created if doesn't already exist

DESCRIPTION

Called by **Updater** function to perform operations on a single feature class at a time based on parameters read from the associated *.ini file.

Reads the associated *.ini file for: data source & target, SQL expression, datum transformation & field mapping assignment. Calls multiple functions in sequence to perform "extract, transform & load" operations. Error messages from "child" functions cause **ReplaceIntData** to send a notification to the **Updater** function, which in turn causes the batch script window to stay open with an error message. If no errors occur, each subsequent function is called, performing various data preparation tasks. This function also sets some environments and repairs geometry.

IMPORTS

ConfigParser	Standard Python module for parsing config (*.ini) files
ConfigLogging	Configures the log files' setup to record the activities of the **ReplaceIntData** function
ValidateFields	Checks matched fields for type and length.
CreateFieldMapping	Using dictionary created by parsing the *.ini file, creates field mapping object for use by the append function
Truncate	Deletes all old data in target feature class in target geodatabase

BuildSql	Builds a where clause/SQL expression with appropriate field delimiters for SOD(s) as specified in the configuration script tool GUI.
AppendData	Appends new data to target feature class in output geodatabase.
UniqueId	Checks for unique ids in SID field.
CalcLatLong	For point layers, re-calculates the latitude/longitude in the appropriate fields.
RemoveUnwantedMetadata	Removes the geoprocessing history and machine name info from the metadata

CONTAINING SCRIPT

replacedata.py

EXTERNAL FUNCTIONS

External functions receive inputs from the **ReplaceIntData** function and (most) return multi-value outputs. The first item in the output is always True or False as a simple indicator of success/failure, and the second is usually a more detailed success/error message. A few functions return additional items. Outcomes are accessed by index, so output[0] is the first value (True/False), output[1] is either the message or other output.

ConfigParser

FUNCTION

ConfigParser(config_file)

INPUTS

config_file	File path/name of current configuration (*.ini) file

DESCRIPTION

Reads configuration (*.ini) file into a parser object.

Parameters are obtained from the parser object with the “get” method and stored in local variables for use by the other functions in the script.

CONTAINING SCRIPT

Standard Python module ConfigParser [in Python 3.x, will change to lowercase “configparser”]

ValidateFields

FUNCTION

ValidateFields(srcFC, destFC, fldMapDict)

INPUTS

inputFeatureClass	Source feature class – new data
outputFeatureClass	Target feature class – old data that are going to be deleted and replaced
fldMapDict	Field map dictionary – a Python dictionary created by the parser as it reads the contents of the [FIELD_MAPPING] section of the associated *.ini file.

DESCRIPTION

Checks matched fields for type and length.

Execution of the function only halts for a source string field that is too long to fit in the target field. Mismatched field types result in a warning message in warnlog, but do not stop processing. Shapefile and SDE field type mismatches seem to be handled appropriately by the append function.

CONTAINING SCRIPT

validatefields.py

CreateFieldMapping

FUNCTION

CreateFieldMapping(inputFC, outputFC, fmDict)

INPUTS

inputFeatureClass	Source feature class – new data
outputFeatureClass	Target feature class – old data to be deleted and replaced
fldMapDict	Field map dictionary – a Python dictionary created earlier by reading in the contents of the [FIELD_MAPPING] section of the associated *.ini file. The field mapping pairs are set using the configuration script tool GUI.

DESCRIPTION

Uses a Python dictionary created by parsing the feature class' associated *.ini file to create a field mapping object for use by the **AppendData** function.

Iterates through input field map dictionary, skipping unmatched fields where a schema field is not matched to a source data field. If the schema field name (key) is not equal to the source data field name (value) in the dictionary (i.e. a pair might be ASN=ALIAS1) send the pair to the field mapper so that the **AppendData** function maps them appropriately. Key/value pairs which have the same name will map automatically in the **AppendData** function.

OUTPUT (WITH AN INDEX OF [2])

fieldMappings	Field mappings object used by the **AppendData** function to update the target feature class

CONTAINING SCRIPT

fieldmappings.py

Truncate

FUNCTION

Truncate(outputFC)

INPUTS

outputFeatureClass	Target feature class – data to be deleted by this function

DESCRIPTION

Deletes all records in the target feature class.

CONTAINING SCRIPT

truncatetable.py

BuildSql

FUNCTION

BuildSql(inputFC, field, value)

INPUTS

inputFeatureClass	Source feature class – new data
sod	SOD field name
srcOfData	Source agency code entered in the configuration script tool GUI and read from the *.ini file (e.g. SANMCO)

DESCRIPTION

Restricts the input of the append operation to only that data stewarded by the agency, excluding any other jurisdictions' data that may be in the source data. For example, 'SOD' = "SANMCO".

If additional parameters are set in the configuration script tool GUI (for stewards of multiple jurisdictions or other parameters), these are appended to the result of the **BuildSql** function in the **ReplaceIntData** function.

OUTPUT (WITH AN INDEX OF [1])

whereClause	e.g. 'SOD'="SANMCO"

CONTAINING SCRIPT

buildsql.py

AppendData

FUNCTION

AppendData(inputFC, outputFC, whereClause, fieldMappings)

INPUTS

inputFeatureClass	Source feature class – new data
outputFeatureClass	Target feature class where new data is to be appended
whereClause	Query parameters created by **BuildSql** & the line after it (i.e. records with the appropriate SOD and additional parameters if specified in the configuration script tool GUI)
fieldMappings	Field mappings object created by **CreateFieldMappings** function (i.e. the field cross-walk)

DESCRIPTION

Appends new data to target feature class.

Uses the SQL expression (a.k.a. where clause) created by **BuildSql** & field mappings object created by **CreateFieldMapping** to append new data to now-empty target feature class. Prior to appending the data, the arcpy.env.geographicTransformations is set to respect the appropriate datum transformation from source data to the target of WGS84. The datum transformation is set in the configuration script tool GUI.

CONTAINING SCRIPT

appenddata.py

UniqueId

FUNCTION

UniqueId(inputFC, idFieldName)

INPUTS

outputFeatureClass	Target feature class – where the new data were just appended
idField	SID – hard coded just above function call

DESCRIPTION

Checks for unique values in SID field.

If SID field doesn't exist, this function is skipped. If non-unique or invalid IDs are found, logs a WARNING in the warnlog.log file. This will not cause execution to halt, however. Warnlog.log needs to be monitored to see if agency has duplicate SID problems.

CONTAINING SCRIPT

uniqueid.py

CalcLatLong

FUNCTION

CalcLatLong(inFC)

INPUTS

outputFeatureClass	Target feature class

DESCRIPTION

For point layers, calculates lat/long into either LAT/LON or LAT_WGS84/LON_WGS84 fields.

CONTAINING SCRIPT

calclatlong.py

RemoveUnwantedMetadata

FUNCTION

RemoveUnwantedMetadata(fgdb, fc)

INPUTS

outputFileGdb	Target file geodatabase
outputFeatureClass	Target feature class

DESCRIPTION

Deletes unwanted data provider geoprocessing history and machine name info from metadata, to elide identifiable information.

CONTAINING SCRIPT

removemetadata.py

ZipFgdb

FUNCTION

ZipFgdb(dataPath)

INPUTS

dataPath	Full path to directory where output geodatabase is located

DESCRIPTION

Deletes old zip file if it exists, then adds all the files in the geodatabase recursively (except *.lock files).

CONTAINING SCRIPT

zipfgdb.py

SendEmail (optional)

FUNCTION

SendEmail(message)

INPUTS

Text of message to be sent

DESCRIPTION

Optional function to provide email notifications.

Needs access to smtp server & credentials, which are set within the function (Python script file = sendmessage.py). Also, sendType variable in **Updater** function must be set to “email”. Could also be configured for text message alerts with a Twilio account.

CONTAINING SCRIPT

sendmessage.py

INTEGRATION SCRIPTS & FUNCTIONS HIERARCHY DIAGRAM

(scriptname.FunctionName)

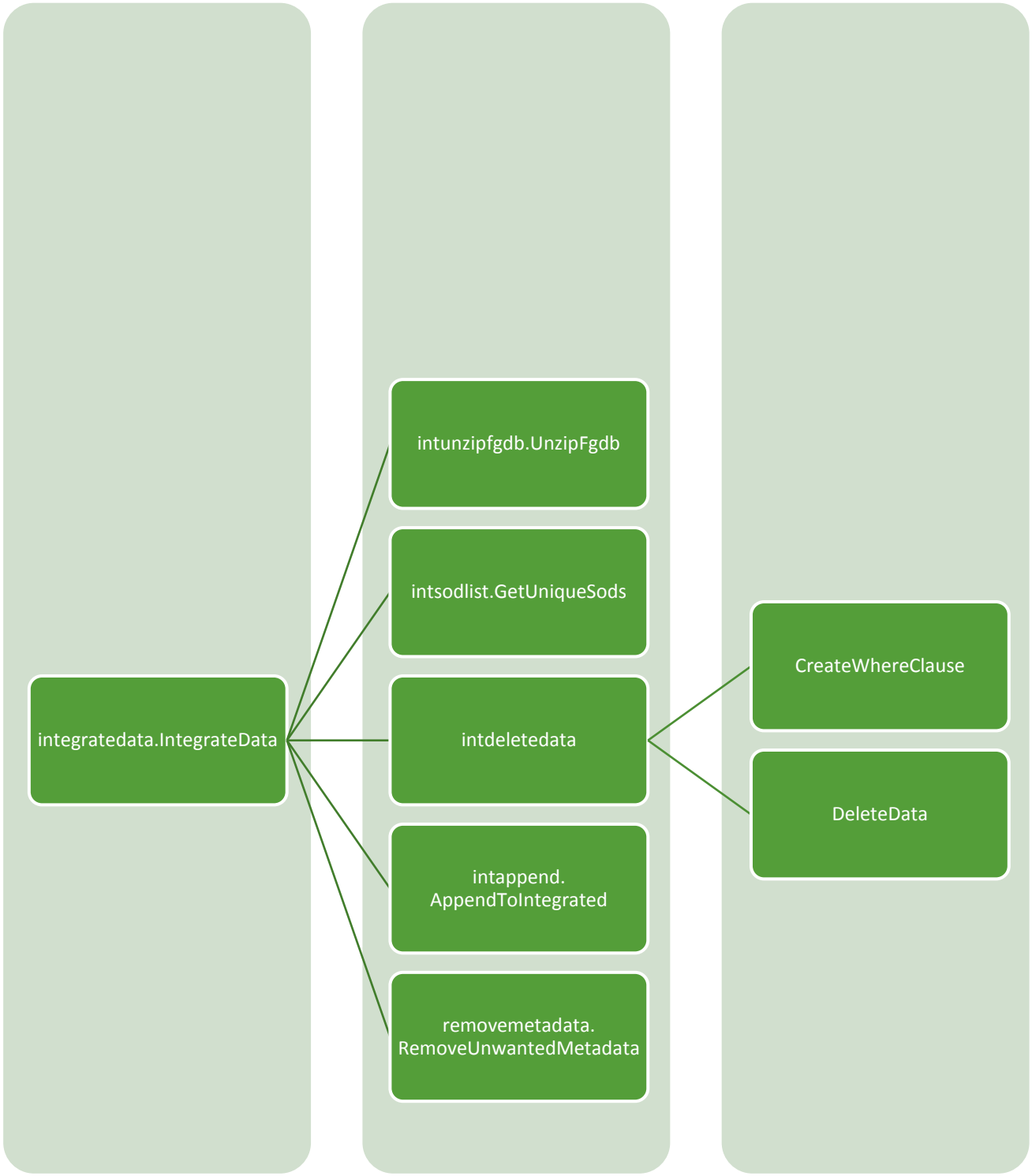


Figure 5 Integration Scripts & Functions Hierarchy

FUNCTIONS

Note: Function inputs are named per the names of the variables in the scripts: the names of inputs vary between calling and executing functions. In other words, what is listed in parentheses might not match what is listed in "inputs".

IntegrateData

FUNCTION

IntegrateData(srcDir, trgtDir, integratedGdb)

INPUTS

srcDir	Source directory – where the source data zip files are located
trgtDir	Directory to which files will be unzipped
integratedGdb	Existing target geodatabase containing feature classes to be updated with new data

DESCRIPTION

Calls functions to unzip all the new source data zip files in a directory & replace the data in an existing file geodatabase. Loops through the unzipped geodatabases, performing the processes one time for each geodatabase.

For each agency geodatabase, gets a Python dictionary of the unique SOD values in each *new* feature class; the key is the feature class name and the value is a list of the valid SODs (see **CreateUniqueSods**). This dictionary is used to create a query to select and delete the appropriate old data. For most agencies, the query will simply contain their agency SOD code. Gunnison County stewards several counties' data which need to be replaced. The valid SOD values vary between feature classes, so their queries will consist of SOD lists with multiple values, different for each feature class.

After the unique SOD dictionary is created, **DeleteData** deletes out the old data which matches. Lastly, the new data are appended with **AppendToIntegrated**. This is all done inside a loop for each geodatabase.

IMPORTS

UnzipFgdb	Unzips source data zip files from one directory into another (specified by tool).
GetUniqueSods	Creates dictionary of unique SOD values.
intdeletedata	(contains 2 functions) Deletes old data from existing integrated file geodatabase based on unique SODs contained in the new data
AppendToIntegrated	Appends new data into existing file geodatabase based on unique SODs dictionary
RemoveUnwantedMetadata	Removes the geoprocessing history and machine name info from the metadata

CONTAINING SCRIPT

integratedata.py

EXTERNAL FUNCTIONS

External functions receive inputs from the **IntegrateData** function and (most) return multi-value outputs. The first item in the output is always True or False as a simple indicator of success/failure, and the second is usually a more detailed success/error message. A few functions return additional items. Outcomes are accessed by index, so output[0] is the first value (T/F), output[1] is either the message or other output.

UnzipFgdb

FUNCTION

UnzipFgdb(srcDir,trgtDir)

INPUTS

srcDir	Source directory containing files to be unzipped
trgtDir	Target directory (preferably a new location, not nested) into which it will unzip geodatabases of the new data.

IMPORTS

os, sys, shutil, zipfile (arcpy not needed)

DESCRIPTION

Deletes old target directory if it exists. Then, extracts the zip files found in the source directory to a re-created or new target directory, as specified in the script tool. Will go into any subdirectories of the source directory looking for zips, so pay attention to nested directories.

CONTAINING SCRIPT

unzipfgdb.py

GetUniqueSods

FUNCTION

GetUniqueSods(fgdb, sodField)

INPUTS

fgdb	Current file geodatabase in loop
sodField	SOD field name (“SOD”)

DESCRIPTION

Creates list of valid SOD values for each feature class in a file geodatabase.

Adds the valid SODs to a dictionary with the feature class name as the key, and the list as the value associated with each key. For example, {STRUCTURES: [GUNNCO, HINSCO, SAGUCO], ROADS: [GUNNCO,MINECO]}. Returns the dictionary.

OUTPUT (WITH AN INDEX OF [2])

fgdbDict	Dictionary of valid SOD values lists per feature class

CONTAINING SCRIPT

intsodlist.py

DeleteData

FUNCTION

DeleteData (intGdb, wcDict)

INPUT

integratedGdb	Existing, integrated geodatabase with data to be deleted
wcDict	Dictionary of valid feature class names/SOD lists for building the where clause

DESCRIPTION

Loops through feature classes in current file geodatabase and deletes all old data.

Calls **CreateWhereClause** function to create a where clause/SQL expression. The where clause/SQL expression is used to create a feature layer and then a selection which is cycled through with a cursor and the matching data deleted.

CONTAINING SCRIPT

intdeletedata.py

CreateWhereClause

FUNCTION

CreateWhereClause(validSodList)

INPUT

validSodList	List of valid SOD values associated with feature class name

DESCRIPTION

Receives the list of valid SOD values for a feature class from the **DeleteData** function. Builds a where clause/SQL expression which contains each of the valid SODs with an OR between them. Returns completed where clause.

OUTPUT

whereClause SQL expression built from valid SOD values (e.g. "'SOD'="GUNNCO" OR 'SOD'="MINECO"")

CONTAINING SCRIPT

intdeletedata.py

AppendToIntegrated

FUNCTION

AppendToIntegrated(fgdb,intGdb)

INPUTS

fgdb Current new geodatabase, as unzipped

integratedGdb Existing, integrated geodatabase

DESCRIPTION

Appends new data to existing integrated file geodatabase. Schemas should already be matched by the source data preparation processes run by the data providers.

CONTAINING SCRIPT

intappend.py

RemoveUnwantedMetadata (same as above)

FUNCTION

RemoveUnwantedMetadata(fgdb, fc)

INPUTS

outputFileGdb Target file geodatabase

outputFeatureClass Target feature class

DESCRIPTION

Deletes unwanted data provider geoprocessing history and machine name info from metadata, to elide identifiable information.

CONTAINING SCRIPT

removemetadata.py

ACKNOWLEDGEMENTS

Some of the logic and scripts were adapted from the Esri Community Addresses solution. Members of the West Region Data Integration group provided patient and invaluable assistance in testing the tools. Matt Goetsch of the City of Montrose reviewed and critiqued this documentation, as well as providing the initial impetus and ongoing support for the project.

This project was undertaken as a capstone for the Master of Geographic Information Systems program at The Pennsylvania State University under the guidance of James Detwiler.