Python Scripting:

Automating Enterprise Geodatabase Maintenance for Texas State Parks

Michael Potts, GISP

Geodatabase Administrator

Email: michael.potts@tpwd.texas.gov

Purposes of Geodatabase Administration

- Provide Current, Authoritative Data To All Users
  - Data Editors: ArcGIS Advanced
  - Data Users: ArcGIS Basic, Map Services
- Prevent Data Loss
- Maximize Geodatabase Efficiency
  - Design
  - Perform Maintenance Tasks Automatically Or On-demand

Purposes of Geodatabase Administration

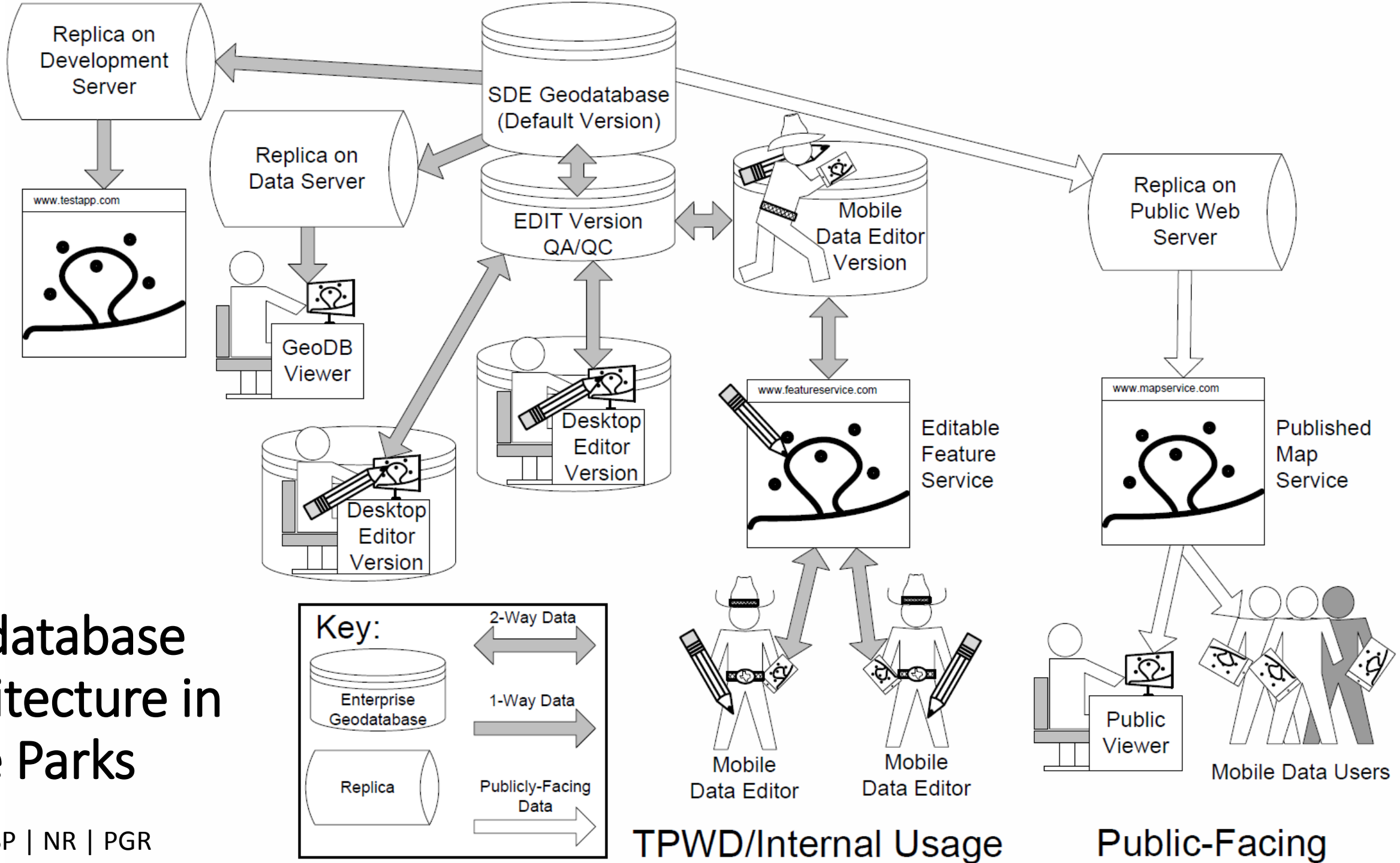
- Ensure Standards, Continuity, And Transferability Of Duties
  - Documentation
  - Global Strategy
- The Database Is Bigger Than Any One Person; It Encompasses The Needs Of The Whole Organization

Purposes of Geodatabase Administration

- Operate behind-the-scenes: ***"things should just work"***
- Is Geodatabase Administration a **thankless** job?
  - If done right, people don't know you've done it
  - You get more *"thank you"s* the more user issues you solve

Assumptions and Environment

- Software
  - ArcGIS (Server) Enterprise 10.5
  - SQL Server (2012) Database
  - Windows Server Environment
  - Python Scripting
- SDE Database Connections And Replicas Must Be Named Consistently
- Servers Are Virtual (No Hardware We Can Access)



Geodatabase Architecture in State Parks

TPWD | SP | NR | PGR

Why Python?

- Great Scripting Tool To Perform Regular Geodatabase Maintenance
- Easy To Learn And Write
- Lots Of Online Support (Official And Unofficial)
- Can Create Advanced Logic To Handle “What-ifs”
- Supported by ArcGIS, ArcGIS Pro, and Standalone Shells
- All ESRI Maintenance Tasks Performed Though The GUI (Graphical User Interface) Can Be Scripted In Python
- Set It And Forget It: Run The Script On A Schedule (Windows Task Scheduler)

https://www.python.org/

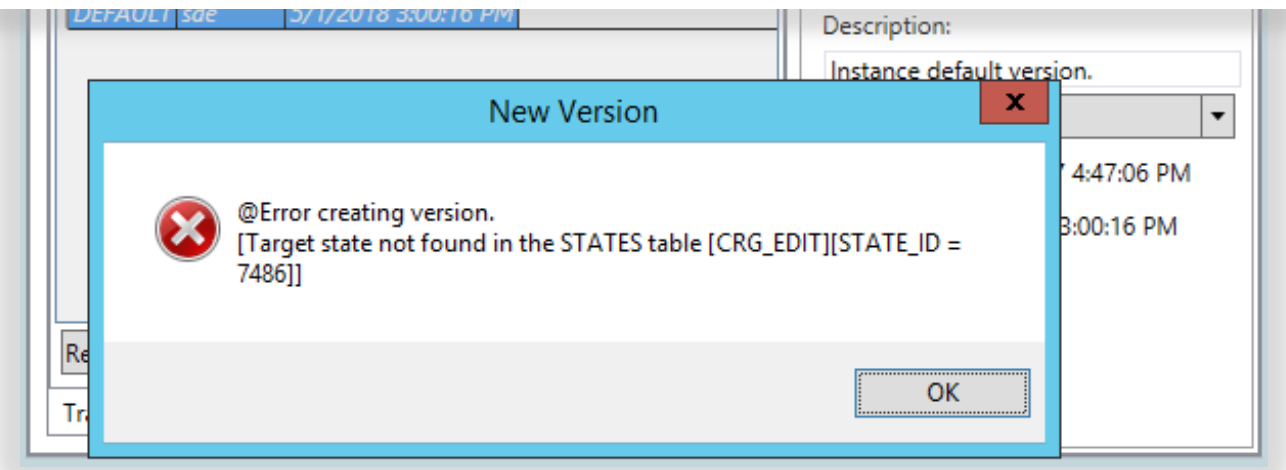
Problems:

- Proper Maintenance Takes Time With GUI Tools
- Model Builder Is Brittle
- Automation Is Easiest If Everything Is Hard-coded
  - Changes In Architecture Are A Headache
  - Non-transportable Without Finding All Data/Machine References
- Once You Make Code, It Looks Like Greek Next Year

Solutions:

- Use The *"**Least Input Variables**"* Rule
  - Anything That Can Be *"Sniffed"*, Should Be *"Sniffed"*
- Break The Code With Geodatabase Structure Permutations
  - If It Only Works One Way, You're Not Done
  - Plan For Differences With Logic
- Track Script Progress And Failure Points With print() Or TextFile.write() Commands
- Handle Your Errors With Gusto
  - You Learn By Fixing Problems
  - Look Up Error Code And Read Blogs
  - Test Multiple Solutions
- Only Hard-code Comments
  - Purpose
  - Logic
  - Lessons Learned Along The Way

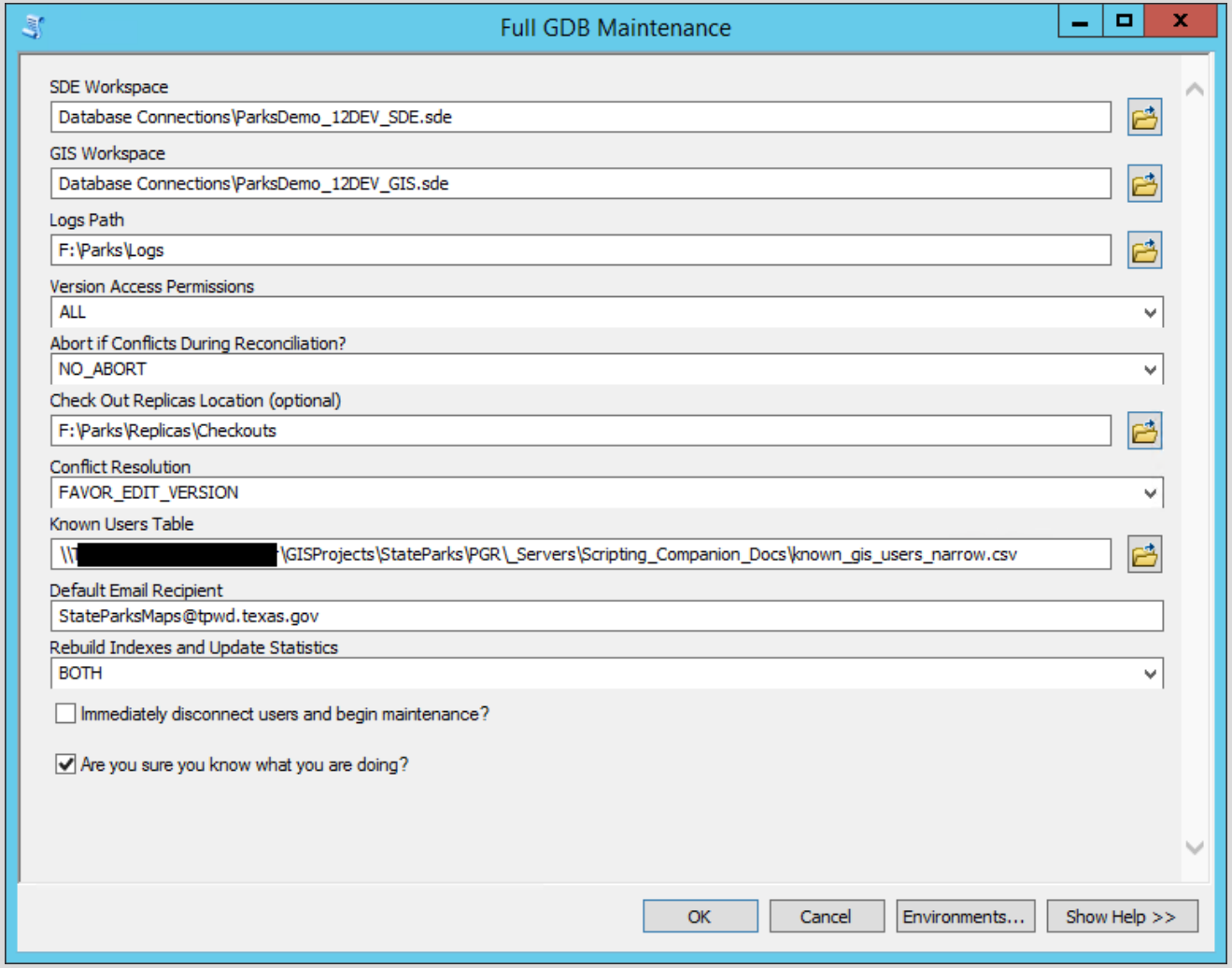
```
Executing: Compress "Database Connections\ParksDemo_12DEV_SDE.sde"
Start Time: Wed Apr 18 08:30:16 2018
ERROR 999999: Error executing function.
Operation Failed [sde.DEFAULT]
Failed to execute (Compress).
Failed at Wed Apr 18 08:30:16 2018 (Elapsed Time: 0.09 seconds)
```



Geodatabase Maintenance Workflow

- Check For Connected Users
- Inform Them To Log Off
- Wait For Them To Log Off
- Sniff Out Versions
- Reconcile Versions
- Sniff Out Replicas
- Synchronize Replicas
- Compress Database
- Recreate Versions
- Rebuild Indexes
- Analyze Datasets
- Inform Any Affected Users To Resume Work

Document And Log
All Steps Undertaken
Send Email Updates



Notification Steps

- Prohibit New Database Connections
- Identify Connected Users
- Perform Identity Checks Against Input User Table
  - **Stop** If A User Is Unknown
  - **Stop** If A Data Owner Is Connected
  - **Continue** If Users Are Known Data Editors Or Data Viewers
- Using Logic, If No Connected Users, Continue Immediately
- Email Connected Users
  - Wait 15 Minutes
- Identify Connected Users Again
  - Email 1 Minute Warning
  - (Actually 2 Minutes)

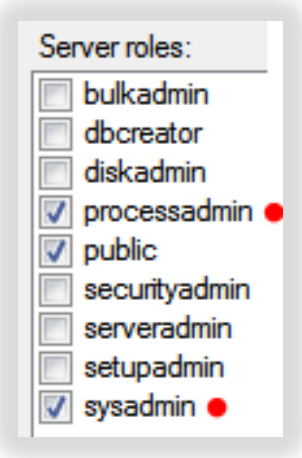
```
arcpy.AcceptConnections(SDE_Workspace, False)

# Build list of connected users
userList = arcpy.ListUsers(SDE_Workspace)
# Pull userNames from userList
userNames = [uL.Name for uL in userList]

# Create two sets to compare, add SDE to cancel out
noMatchUserSet = set(["SDE"]) ## Connected users
matchingUserSet = set(["SDE"]) ## Known users

# Open CSV containing known users; load into list
UserCSV = open(UserCSVPath, 'r')
uCSV = list(csv.reader(UserCSV)) ## Input Variable

# Examine each connected user and match info from CSV
for uN in userNames:
    # Add connected users to noMatchUserSet
    noMatchUserSet.add(uN.strip())
    for uC in uCSV:
        # If UserNames match (user is known)
        if uN == uC[0]:
            # UserName added matchingUserSet
            matchingUserSet.add(uC[0])
            # UserEmail added to emailList
            emailList.append(uC[1])

noMatchUserSet.difference_update(matchingUserSet)
# noMatchUserSet is empty if all users are known
```

Disconnecting Users Requires Membership

- SQL Server Roles
  - “processadmin”: Can Disconnect Users And End Processes In The Server Instance
  - “securityadmin” Can Assign Server-level Permissions And Most Other Actions
  - “sysadmin”: Can Do Anything
- We Want To Disconnect Users To Prevent Data Loss And Achieve Max Compression

```
# This requires SQL processadmin privileges
arcpy.DisconnectUser(workspace, "ALL")
```

SQL Server Management Studio Server Roles for SDE User



Reconcile/Post Without Hard-Coding

- Sniff Out
  - Version Names
  - Version Permissions
  - Version Owners
  - Version Parents
  - Version Children
  - Version Rank `len(version.ancestors)` Must Reconcile/Post In Correct Sequence
    - 0: Default Version
    - 1: Child Of Default
    - 2: Grandchild Of Default
    - 3: Great Grandchild Of Default
- Handle Escape Characters Common In “Domain\username” Scenarios
- Python’s Propensity To Turn “\” Into “\\” Into “\\\\”
- Document Versions As-is For Recreation
  - If A User Owned A Version, Ownership Will Be Appropriated By GIS Dataowner

Sniff out User Versions

- Build a List of Versions
- Version Permissions
  - Public
  - Private
  - Protected
- Create Tuples (Structured List Item)
  - Number Of Children
  - Access Permission

```
# Using logic, build lists of versions by access permissions
for ver in arcpy.da.ListVersions(workspace):
    verParent = str(ver.parentVersionName)
    verName = str(ver.name)
    verAccess = str(ver.access)
    # Rank determines the order of version recreation
    verRank = str(len(ver.ancestors))

    # Number of children parent has
    verChildren = len(ver.children)
    # \/ Build tuples to hold the structured info \/
    # Only grab the versions that are parents
    if verChildren > 0:
        prntChildRank.append((str(verRank), ver.name, str(ver.access)))
    masterVersionList.append((verRank,verParent,verName,verAccess))
    parentChildList.append((verParent,verName,verAccess))

    # Build the PUBLIC_ONLY tuples
    if versionAccess == "PUBLIC_ONLY": ## Input Variable
        if verAccess == "Public": ## Version Property
            publicOnlyVersionList.append((verRank,verParent,verName,verAc
    if verAccess != "Public":
        private_protectedVersionList.append((verRank,verParent,verName,ve
```

“PUBLIC_ONLY” Version Logic Intact, But Tuple Intentionally Trimmed For Brevity

Document The Versions As-IS In A .CSV

- The Script Will Delete The Versions During Reconcile/Post
- If The Script Fails, Having A CSV Of The Versions Helps To Reconstruct Them
- CSV Outputs To Disk, Can Be Manipulated, And Fed Back Into Python
- CSVs Can Contain Anything In Any Order
- Easy To Create On-The-Fly
- East To Access And Navigate Later

```
nmb = 1 ## To have an index number in the log file
# Create and write to the VersionRestoration CSV
with open(VersionRestoration_csv, "w") as versionRestCSV:
    fieldnames = ["in_workspace","parent_ver","ver_name",
                  "permission","rank","owner","new_owner"]
    writer = csv.DictWriter(versionRestCSV, fieldnames=fieldnames,
                            lineterminator = "\n")
    #writer.writeheader() # Don't need the header, but this would
    for item in masterVersionList:
        vs = str(item)
        verRank = str(vs.split("'")[1]).strip()
        verParent = str(vs.split("'")[3]).strip()
        verName = str(vs.split("'")[5]).strip()
        verOwner = str(verName.split(".")[0]).strip()
        plainName = str(verName.split(".")[1]).strip()
        verAccess = str(vs.split("'")[7]).strip()
        # This logic decides the connection to use in
        # rebuilding versions by identifying the owner.
        if verOwner == "sde" or verOwner == "SDE":
            conxn = SDE_Workspace
            newOwner = "sde"
        elif verOwner == "GIS" or verOwner == "gis":
            conxn = GIS_Workspace
            newOwner = "GIS"
        elif verOwner != "GIS" or verOwner != "gis"
            or verOwner != "sde" or verOwner != "SDE":
            masterLog.write("\n\rVersion: " + str(verOwner) +
                    " will be appropriated by GIS user\n\r")
            conxn = GIS_Workspace
            newOwner = "GIS"
        masterLog.write(str(nmb) + ": " + str(item)+"\n\r")
        nmb += 1 ## Iterate index number
        # Command to write the row in the CSV
        writer.writerow({"in_workspace":conxn,"parent_ver":verParent,
        "ver_name":plainName,"permission":verAccess,"rank":verRank,
        "owner":verOwner,"new_owner":newOwner})
```

Create Temp CSV Of Parents: Children

- If Script Has An Error, The Temp Documents Will Persist
  - Troubleshooting Source Of Error (A Name May Contain A Sequence Of Characters That Trouble Python)
    - \a, \b, \f, \n, \r, \t, \v Are Escape Sequences
    - Domain\username Is A Common Format For OS Users
    - “Domain\nsmith” Interprets As
      - ‘Domain’
      - ‘smith’
  - Help Establishing Fail Point

```
# Create and write the ParentChild_csv and populate the parentSet
with open(ParentChild_csv, "w") as parentChildCSV:
    fieldnames = ["parent", "child", "access"]
    writer = csv.DictWriter(parentChildCSV,fieldnames=fieldnames,
                                    lineterminator="\n")
    for item in parentChildList:
        vs = str(item)
        verParent = str(vs.split("'")[1]).strip()
        verChild = str(vs.split("'")[3]).strip()
        verAccess = str(vs.split("'")[5]).strip()
        writer.writerow({"parent":verParent,"child":verChild,
                         "access":verAccess})
        # Sets can only contain 1 of each item
        parentSet.add(verParent)
if versionAccess == "PUBLIC_ONLY":
    masterLog.write("\n\r\n\r" + str(datetime.datetime.now())+
                    "\n\rpublicOnlyVersionList is below:\n\r")
    nmb = 1
    for item in publicOnlyVersionList:
        masterLog.write(str(nmb) + ": " + str(item)+"\n\r")
```

“PUBLIC_ONLY” Version Logic Intentionally Trimmed For Brevity

Create Dictionary Of Parents: Children

- Need To Populate Dictionary Of Parents: Children
- Dictionaries Are Great For 1: Many Relationships
- Easy to create and call later

```
versionTreeDict = {}
tempChildList = []
# Loop through parents
for p in parentSet:
    PCCSV = open(PCCSVPath)
    parentChildCSV = csv.reader(PCCSV)
    # Loop through parent/child pairs
    for row in parentChildCSV:
        verParent = str(row[0])
        verChild = str(row[1])
        # If current child has the same parent
        # add to tempChildList
        if p == verParent:
            tempChildList.append(verChild)
    # Define the parent equal to all its children
    versionTreeDict[p] = tempChildList
    # Clear tempChildList
    tempChildList = []
    PCCSV.close()
# Add to logs
masterLog.write("\n\r" + str(datetime.datetime.now())+ "\n\rVersio
nmb = 1
for item in versionTreeDict:
    masterLog.write(str(nmb) + ": " + str(item) + ":" + str(versio
    nmb += 1

# Begin publicVersionTreeDict creation
if versionAccess == "PUBLIC_ONLY":
```

"PUBLIC_ONLY" Version Logic Intentionally Trimmed For Brevity

OS User Versions Can Become “Dirty”

- Thus Begins The Loop To Reconcile\Post
- If Domain\user Becomes Domain\\\\user, Must Clean It Up
- If There Were No Dirty Children, Just Use Original verChildren

```
for v in prntChildRank:
    cleanChildren = [] ## Clear out list
    verParentTuple = str(v) ## parent, child
    verParent = str(verParentTuple.split("'")[3])
    verChildren = versionTreeDict[verParent]
    if versionAccess == "PUBLIC_ONLY":
        verChildren = publicVersionTreeDict[verParent]
    if "\\\\" in verParent: ## Loop to clean parent
        whittle = verParent
        verParent = str(whittle.split("\\")[0])+"\\"+str(whittle.split("\\")[-1])
    for c in verChildren: ## Loop to clean children
        if "\\\\" in c:
            whittle = c
            cc = str(whittle.split("\\")[0])+"\\"+str(whittle.split("\\")[-1])
            cleanChildren.append(cc)
        else: ## Else: a child is already clean, append to list
            cleanChildren.append(c)
    if len(cleanChildren) > 0: ## Else: use verChildren from above
        verChildren = []
        for c in cleanChildren:
            verChildren.append(c)
```

Finally Run GP Tool Within Loop From Last Slide

```
        verAccess = str(verParentTuple.split("'")[5])
        if versionAccess == "PUBLIC_ONLY":
            if verAccess == "Public":
                arcpy.ReconcileVersions_management(workspace,"ALL_VERSIONS",
                    verParent,verChildren,"LOCK_ACQUIRED",RecPostConflictHandling,
                    "BY_OBJECT",ConflictResolution,"POST","DELETE_VERSION",
                    TempReconcileLog_txt)
                masterLog.write(str(nmb)+": "+"Parent: "+verParent+"; Children: "+
                    str(verChildren)+" just reconcile/posted\n\r")
                input = open(TempReconcileLog_txt, "r")
                for line in input:
                    recPostLog.write(line)
                nmb += 1
                input.close()

        elif versionAccess == "ALL":
            arcpy.ReconcileVersions_management(workspace,"ALL_VERSIONS",verParent,
                verChildren,"LOCK_ACQUIRED",RecPostConflictHandling,"BY_OBJECT",
                ConflictResolution,"POST","DELETE_VERSION",TempReconcileLog_txt)
            masterLog.write(str(nmb)+": "+"Parent: "+verParent+"; Children: "+str(verChildren)+
                " reconcile/posted at: "+str(datetime.datetime.now())+"\n\r")
            input = open(TempReconcileLog_txt, "r")
            for line in input:
                recPostLog.write(line)
            nmb += 1
            input.close()

    recPostLog.write("\n\rAt: " + str(datetime.datetime.now()) +
                ", finished reconciling and posting\n\r")
```

Automatically Synchronize Replicas

- Using Data Access Module, Can Sniff
  - Replica Name
  - Type (1-Way, 2-Way, Check-Out)
  - Owner
  - Parent Version
  - ~~Replica Location or SDE Connection File~~
- Requires Predictable Database Connection Naming
  - Database_Server_User.sde
- Requires Standard ArcGIS Installation with SDE connection files in the normal location
- Export Replica Schema (XML) As Temp File
  - 1-Way
  - 2-Way
- Read XML To Create Dictionary Of reptogdbDict[repname] = gdb
- Check-Out Replicas
  - Must Provide Location (They Are Expected To Move Around)
  - Must have “CheckOut” In Their Name
- Synchronize Replicas Once All is Known

Sniffing Replicas

- Use Data Access Module For Initial Discovery
- Write To Log Files
- Create XML Files

```
# Sniff out replicas
replicaVersions = arcpy.da.ListReplicas(SDE_Workspace)
Count = len(replicaVersions)
# Write the total number of replicas
ReplicaDiscoSync.write("\n\r" + str(SDE_Workspace) + " has "
                       + str(Count) + " replicas\n\r")
for rep in replicaVersions:
    # If Check-Out, create tuple of useful info
    if rep.type == "CheckOut":
        checkoutVerList.append((rep.name, rep.type, rep.owner, rep.version))
    # Export replica schema for 1 and 2 way replicas
    else:
        arcpy.ExportReplicaSchema_management(SDE_Workspace, Logs_path +
                          "/TempRepSchema_" + rep.name + "_.xml", rep.name)
        # Export replica schema for 1 and 2 way replicas
        ReplicaDiscoSync.write("\n\rExported: " + str(Logs_path +
                             "/TempRepSchema_" + rep.name + "_.xml"))
        # Create list of XML files for later perusal
        xmlList.append((rep.name, Logs_path + "/TempRepSchema_" + rep.name + "_.xml"))
```

Read XML Files For 1-Way Replicas

- XML File Provides File Geodatabase Physical Location
- Add To Dictionary

```
# Search through replica schemas created above
ReplicaDiscoSync.write("\n\r\n\rExamining XML files for 1 and 2 way replica gdb locations:")
nmb = 0
for doc in xmlList:
    nmb += 1
    repname = str(doc[0])
    path = str(doc[1])
    tree = ET.parse(path)
    root = tree.getroot()
    for accs in root.iter('AccessType'):
        typeString = accs.text
        ReplicaDiscoSync.write("\n\rEntering logical sequence " + str(nmb) + ":")
        # one-way replica
        if typeString == "esriReplicaChildReadOnly":
            # XML tag holding conxn info
            for cons in root.iter('SibConnectionString'):
                dbString = cons.text
                # Slice out the gdb name from the property tags
                gdb = dbString.split("=")[1].split(";")[0]
                reptogdbDict[repname] = gdb
                ReplicaDiscoSync.write("\n\rType: esriReplicaChildReadOnly\n\r" +
                    "reptogdbDict" + "[" + str(repname) + "] = "  + str(gdb) + "\n\r")
```

Read XML Files For 2-Way Replicas

- Find
  - Database Username
  - Database Name

```
# two-way replica; find sde connection file
elif typeString == "esriReplicaBothReadWrite":
    possResult = []
    for cons in root.iter('SibConnectionString'):
        dbString = cons.text
        # returns server name with excess lard
        servSearch = dbString.split(";")[1].split("=")[1]
        if "-" in servSearch:
        # returns server name; the last split accounts hyphen
        # after which the server name is used; not ideal naming!
        nservSearch = servSearch.split("-")[1]
            servSearch = nservSearch ## switch the values back
        # returns SQLDB name
        dbName = dbString.split(";")[5].split("=")[1]
        # returns user
        usrSearch = dbString.split(";")[6].split("=")[1].upper()
```

Read XML Files For 2-Way Replicas

- Using The Username And Database Name, Find the SDE Connection File On Local Machine
- Add The Connection File Location To The Dictionary

```
# Find the SDE connection file on local machine
sde_dir_comp = os.path.join(os.getenv("APPDATA"), r"ESRI")
connDir = os.listdir(sde_dir_comp)
for folders in connDir:
    if "Desktop" in folders:
        possResult.append(str(folders))
# If more than one installation of ArcGIS,this sorts results
possResult.sort(reverse=True)
# picks most recent (highest) from sorted results
conxnFolder = possResult[0]
# Join path
sde_path = os.path.join(os.getenv("APPDATA"),
                        r"ESRI",conxnFolder,r"ArcCatalog")
sde_results = os.listdir(sde_path)
# Loop through SDE connections for right one
for item in sde_results:
    #if  in item and "SDE" in item:
    if dbName in item and usrSearch in item:
        actualSDEConn = item
        gdb = os.path.join(sde_path, item)
        reptogdbDict[repname] = gdb
        ReplicaDiscoSync.write("\n\rType: esriReplicaBothReadWrite\n\r" +
            "reptogdbDict" + "[" + str(repname) + "] = "  + str(gdb) + "\n\r")
```

Check Out Replicas

- Find Their Physical Location From The User Input
- Add The Path To The Dictionary

```
# CheckOutDir is user input
CheckOut_results = os.listdir(CheckOutDir)
for gdbs in CheckOut_results:
    if "CheckOut" in gdbs:
        repname = gdbs.split(".")[0]
        path = os.path.join(str(CheckOutDir), str(gdbs))
        checkOutGDBList.append(str(path))
        reptogdbDict[repname] = path # repname will be missing owner!
        ReplicaDiscoSync.write("\n\rType: CheckOut Replica\n\r" +
                "reptogdbDict" + "[" + str(repname) + "] = "  +
                str(path) + "\n\r")
```

Finding geodatabase_1

- Earlier, We Found The SDE Connections On The Local Machine
- geodatabase_1 Is The Parent Geodatabase For The Replica, But The Data Access Module Only Told Us The Owner
- Use The Owner To Find The Right SDE Connection String

```
# Using owner, return the DB connection string for geodatabase_1
for rep in replicaVersions:
    usrSearch = rep.owner
    replicaName = rep.name
    possResult = []
    conxnPosses = []
    dbName = (SDE_Workspace.split("_")[0]).split("\\")[1]
    # sde_results was created in a previous step; is every
    # SDE connection string on the local machine
    for item in sde_results:
        if dbName in item and usrSearch in item:
            actualSDEConn = item
            gdb = os.path.join(sde_path, item)
            conxnPosses.append((len(gdb),gdb))
    if len(conxnPosses) > 1:
        conxnPosses.sort()
        nmb = 0
        for item in conxnPosses:
            nmb += 1
        geodatabase_1 = conxnPosses[0]
```

Create Tuples For GP Tool

- Each tuple added to replicaList will contain the necessary inputs for GP tool
- Use logic to grab appropriate input parameters

```
# Translate rep type into GP tool input
if rep.type == "OneWay":
    ReplicaDiscoSync.write("\n\rGeoDBName: " + (reptogdbDict[rep.name]) + "\n\r")
    replicaType = "FROM_GEODATABASE1_TO_2"
    replicaList.append((replicaName, reptogdbDict[rep.name], replicaType))
elif rep.type == "TwoWay":
    ReplicaDiscoSync.write("\n\rGeoDBName: " + (reptogdbDict[rep.name]) + "\n\r")
    replicaType = "BOTH_DIRECTIONS"
    replicaList.append((replicaName, reptogdbDict[rep.name], replicaType))
elif rep.type == "CheckOut":
    repname = rep.name.split(".")[1]
    replicaType = "FROM_GEODATABASE2_TO_1"
    ReplicaDiscoSync.write("\n\rGeoDBName: " + (reptogdbDict[repname]) + "\n\r")
    replicaList.append((replicaName, reptogdbDict[repname], replicaType, geodatabase_1[1]))
```

Run GP TOOL Synchronize Changes: 2-Way

```
ReplicaDiscoSync.write("\n\rReplica GP Tool Inputs:\n\r")
nmb = 0
for rep in replicaList:
    nmb += 1
    in_replica = str((rep[0]).split(".")[1])
    geodatabase_2 = str(rep[1])
    in_direction = str(rep[2])
    geodatabase_1 = SDE_Workspace
    conflict_policy = "IN_FAVOR_OF_GDB1"
    conflict_definition = "BY_OBJECT"
    reconcile = "DO_NOT_RECONCILE"
    #if SyncBox == "true" and Surety == "true":
    if in_direction == "BOTH_DIRECTIONS":
        if SyncBox == "true" and Surety == "true":
            arcpy.SynchronizeChanges_management(geodatabase_1, in_replica, geodatabase_2,
                in_direction, conflict_policy, conflict_definition, reconcile)
            ReplicaDiscoSync.write("   Replica: " + in_replica + " successfully synchronized\n\r")
```

Run GP TOOL Synchronize Changes: 1-Way And Check Out

```
elif in_direction == "FROM_GEODATABASE1_TO_2":
    ReplicaDiscoSync.write(str(nmb) + ": " + geodatabase_1 + "\n\r    " + in_replica + "\n\r    " + geod
    ReplicaDiscoSync.write("    " + conflict_policy + "\n\r    " + conflict_definition+ "\n\r    " + reco
    if SyncBox == "true" and Surety == "true":
        arcpy.SynchronizeChanges_management(geodatabase_1, in_replica,
            geodatabase_2, in_direction, '', '', '')
        ReplicaDiscoSync.write("   Replica: " + in_replica + " successfully synchronized\n\r")
elif in_direction == "FROM_GEODATABASE2_TO_1":
    geodatabase_1 = str(rep[3])
    reconcile = "RECONCILE " ## Space required for the gp tool input
    conflict_policy = "IN_FAVOR_OF_GDB2" # for checkouts!
    ReplicaDiscoSync.write(str(nmb) + ": " + geodatabase_1 + "\n\r    " + in_replica + "\n\r    " + geod
    ReplicaDiscoSync.write("    " + conflict_policy + "\n\r    " + conflict_definition+ "\n\r    " + reco
    if SyncBox == "true" and Surety == "true":
        arcpy.SynchronizeChanges_management(geodatabase_1, in_replica, geodatabase_2,
                        in_direction, conflict_policy, conflict_definition, reconcile)
        ReplicaDiscoSync.write("   Replica: " + in_replica + " successfully synchronized\n\r")
        # This deletes the checkout replica geodb after syncing
        arcpy.Delete_management(geodatabase_2)
        ReplicaDiscoSync.write("Deleted checkout replica: " + str(geodatabase_2) + "\n\r")
```

Logging Intentionally Cut Off For Brevity

Compress the Database

- Compress Using The Database Administrator Connection (User Input)

```
toolStart = datetime.datetime.now()
comp_log.write("\n\rStarting tool: arcpy.Compress_management.\n\r")
arcpy.Compress_management(SDE_Workspace)
toolEnd = datetime.datetime.now()
masterLog.write("\n\r" + "Compression tool elapsed time: " + str(toolEnd - toolStart))
comp_log.write("\n\r" + "Compression tool elapsed time: " + str(toolEnd - toolStart) + "\n\r")
```

Check The Level Of Compression

- Create A Table View Of “sde.SDE_Versions”
- Export To TempCSV
- Read CSV
- Sort By State
- Write To Log

```
Starting tool: arcpy.Compress_management.

Compression tool elapsed time: 0:00:04.459000

Checking state tree:
0,DEFAULT
0,SYNC_SEND_3513_171
0,SYNC_SEND_5122_102
0,SYNC_SEND_4722_104

Largest state pair: 0,SYNC_SEND_4722_104

Current state: 0
```

```
# Checking on state tree
inTable = SDE_Workspace + "\sde.SDE_versions"
out_table = "VersionsTable"
out_path = Logs_path
out_name = r"\TempVersTablePostCompress.csv"
arcpy.MakeTableView_management(in_table=inTable, out_view=out_table)
arcpy.TableToTable_conversion(out_table, out_path, out_name)
versionCSV = open(out_path + out_name)
vrsCSV = csv.reader(versionCSV)
stateList = []
largest = ""
comp_log.write("\n\rChecking state tree:")
for row in vrsCSV:
    stateID = str(row[5]).split(".")[0]
    verName = str(row[1])
    if stateID != "state_id":
        stateList.append((stateID + "," + verName))
        comp_log.write("\n\r" + ((stateID + "," + verName)))
versionCSV.close()
stateList.sort(reverse=True)
currStatePair = stateList[1]
comp_log.write("\n\r\n\rLargest state pair: " + currStatePair)
```

Restore User Versions

- Begin Accepting Connections Again
- Check Existing Versions
- Compare Existing Versions To Initial Input Versions CSV
- Recreate What Is Missing

Prepare For Restoration

- Begin Accepting Connections To Database
- Create Several Sets And Lists Used For Comparison
- Sniff Out Existing Versions
- Add Existing Versions To A Set

```
workspace = SDE_Workspace
arcpy.env.workspace = workspace
arcpy.AcceptConnections(workspace, True)
verRest.write("\n\rAccepting connections again")
existingVersionSet =  set([])
supposedVersionSet =  set([])
createVersionList = []
supposedVersionList = []
for ver in arcpy.da.ListVersions(workspace):
    splitItem = ver.name
    splitItems = splitItem.split(".")
    plainName = str(splitItems[1])
    existingVersionSet.add(plainName)
```

Decide What To Restore

- Open CSV Created Before Reconcile/Post
- Add Versions To A Supposed Versions Set
- Compare Existing Versions To Supposed Versions
- Add Versions To Create To Create Version List

```
vcCSV = open(vcCSVPath)
version_Mod_Cre = csv.reader(vcCSV)
for row in version_Mod_Cre:
    supposedVersionSet.add(row[2])
    supposedVersionList.append(row)
verRest.write("\n\rexistingVersionSet:\n\r")
nmb = 1
for ex in existingVersionSet:
    verRest.write(str(nmb) + ": " + str(ex) + "\n\r")
    nmb += 1
verRest.write("\n\rsupposedVersionSet:\n\r")
nmb = 1
for sup in supposedVersionSet:
    verRest.write(str(nmb) + ": " + str(sup) + "\n\r")
    nmb += 1
supposedVersionSet.difference_update(existingVersionSet)
for item in supposedVersionSet:
    createVersionList.append(item)
```

What If There's Nothing To Restore?

- Check The Length Of Versions To Create
- Skip To Indexing If Nothing
- Write To Log What Will Be Created

```
versToCreate = len(supposedVersionSet)
if versToCreate == 0:
    masterLog.write("\n\rAll versions exist; skipping to indexing")
    verRest.write("\n\rAll versions exist; skipping to indexing")
elif versToCreate > 0:
    verRest.write("\n\rCreateVersionList:\n\r")
    nmb = 1
    for sup in createVersionList:
        verRest.write(str(nmb) + ": " + str(sup) + "\n\r")
        nmb += 1
```

Run GP Tool Create Version

- Loop Through Input Lists And Tuples
- If Necessary, Data Owner Will Hijack Version From Domain User
- Create Versions
- Write To Log

```
nmb = 1
masterLog.write("\n\rBeginning to create versions in a loop.")
verRest.write("\n\rBeginning to create versions in a loop.")
for row in supposedVersionList:
    for cre in createVersionList:
        if cre == row[2]:
            in_workspace = str(row[0])
            parent = str(row[1])
            # Compare original Owner with GeoDB DataOwner
            if str(row[5]) != str(row[6]):
                # Replace DataOwner string with original Owner
                parent = str(row[6])+"."+str(row[1]).split(".")[1]
            version_name = str(row[2])
            access_permission = str(row[3])
            arcpy.CreateVersion_management(in_workspace, parent,
                                           version_name, access_permission)
            verRest.write("\n\r" + str(nmb) + ": Created version: " +
                version_name + " at: " + str(datetime.datetime.now()))
            nmb += 1
```

Rebuild Indexes And Analyze Datasets

- Both Improve Performance Of Geodatabase
- Must Sniff Out All Data In Geodatabase For Both

```
dataList = arcpy.ListTables() + arcpy.ListFeatureClasses() + arcpy.ListRasters()
```

- Rebuild Indexes First
  - Do System Tables First (Fast)
  - Do Datasets Second (Very Slow)
- Analyze Datasets Second
  - Do System Tables First (Fast)
  - Do Datasets Seconds (Slow)

Rebuild Indexes

- Do the system Tables First
- Datasets Second
- Write To Log Files Along With Elapsed Time

```
nmb = 0
if indexAnalyze == "REBUILD_INDEXES_ONLY" or indexAnalyze == "BOTH":
    # Rebuild Indexes first for system tables (as DB Owner)
    # then on datasets as data owner (GIS)
    indSys = datetime.datetime.now()
    arcpy.RebuildIndexes_management(SDE_Workspace,"SYSTEM","","ALL")
    indSys2 = datetime.datetime.now()
    nmb += 1
    IndexAnalyzeLog.write("\n\r" + str(nmb) + ": Rebuilt Indexes SYSTEM for ta
    indData = datetime.datetime.now()
    arcpy.RebuildIndexes_management(GIS_Workspace,"NO_SYSTEM",dataList,"ALL")
    indData2 = datetime.datetime.now()
    nmb += 1
    IndexAnalyzeLog.write("\n\r" + str(nmb) + ": Rebuilt indexes for datasets
```

Logging Intentionally Cut Off For Brevity

```
1: Rebuilt Indexes SYSTEM for tables first; Total elapsed time: 0:00:04.967000
2: Rebuilt indexes for datasets (NO_SYSTEM tables); Total elapsed time: 0:17:17.278000
```

Analyze Datasets

- Do the system Tables First
- Datasets Second
- Write To Log Files Along With Elapsed Time

```
if indexAnalyze == "ANALYZE_ONLY" or indexAnalyze == "BOTH":
    # Analyze Datasets updates statistics on system tables (as DB Owner),
    # then on datasets as (data owner)
    analSys = datetime.datetime.now()
    arcpy.AnalyzeDatasets_management(SDE_Workspace, "SYSTEM", "", "ANALYZE_BASE",'
    analSys2 = datetime.datetime.now()
    nmb += 1
    IndexAnalyzeLog.write("\n\r" + str(nmb) + ": Analyzed System tables; Total ela
    masterLog.write("\n\r" + str(nmb) + ": Analyzed System tables; Total elapsed t
    analData = datetime.datetime.now()
    arcpy.AnalyzeDatasets_management(GIS_Workspace, "NO_SYSTEM", dataList, "ANALYZ
    analData2 = datetime.datetime.now()
    nmb += 1
    IndexAnalyzeLog.write("\n\r" + str(nmb) + ": Analyzed Datasets; Total elapsed
    masterLog.write("\n\r" + str(nmb) + ": Analyzed Datasets; Total elapsed time:
```

Logging Intentionally Cut Off For Brevity

```
127: ParksDemo.GIS.ArchSitesPT
128: ParksDemo.GIS.NewExampleFC
129: ParksDemo.GIS.AccessPointLabels
130: ParksDemo.GIS.Topo5Ft
131: ParksDemo.GIS.BoundaryProvisionalPy

1: Rebuilt Indexes SYSTEM for tables first; Total elapsed time: 0:00:04.967000
2: Rebuilt indexes for datasets (NO_SYSTEM tables); Total elapsed time: 0:17:17.278000
3: Analyzed System tables; Total elapsed time: 0:00:04.907000
4: Analyzed Datasets; Total elapsed time: 0:03:56.478000
```

You Could Skip Index And Analyze

- Sometimes It's Not Worth The Wait

```
if indexAnalyze == "NEITHER":
    IndexAnalyzeLog = open(IndexAnalyze_txt, 'a')
    IndexAnalyzeLog.write("\n\rNeither indexes were rebuilt r
    masterLog.write("\n\rNeither indexes were rebuilt nor wer
    IndexAnalyzeLog.close()
```

Logging Intentionally Cut Off For Brevity

```
================================================================================
Index & Analyze option(s): NEITHER. Beginning at 2018-05-01 15:29:16.645000
================================================================================

Index and Analyze beginning with option: NEITHER

Neither indexes were rebuilt nor were statistics updated (Analyze) at: 2018-05-01 15:29:16.645000

================================================================================
Index & Analyze option(s): NEITHER. Completed at 2018-05-01 15:29:16.645000; Total elapsed time: 0:00:00
================================================================================
```

Clean Up Temp Files

```
masterLog.write("\n\rDeleting temp item(s):")
dirFiles = os.listdir(Logs_path)
nmb = 1
for temp in dirFiles:
    if "Temp" in temp:
        os.remove(Logs_path + "\\" + temp)
        masterLog.write("\n\r" + str(nmb) + ": Deleted file " + I
        nmb += 1
    elif "info" in temp:
        shutil.rmtree(Logs_path + "\\" + temp)
        masterLog.write("\n\r" + str(nmb) + ": Deleted directory
        nmb += 1
masterLog.close()
output.close()
```

Logging Intentionally Cut Off For Brevity

```
Deleting temp item(s):
1: Deleted directory F:\Parks\Logs\ParksDemo_Logs\PythonToolBox_Logs\NewLogs20180501\info
2: Deleted file F:\Parks\Logs\ParksDemo_Logs\PythonToolBox_Logs\NewLogs20180501\TempParentChild_ParksDemo_12DEV_SDE.csv
3: Deleted file F:\Parks\Logs\ParksDemo_Logs\PythonToolBox_Logs\NewLogs20180501\TempReconcileLog_ParksDemo_12DEV_SDE.txt
4: Deleted file F:\Parks\Logs\ParksDemo_Logs\PythonToolBox_Logs\NewLogs20180501\TempReconcileLog_ParksDemo_12DEV_SDE.txt.xml
5: Deleted file F:\Parks\Logs\ParksDemo_Logs\PythonToolBox_Logs\NewLogs20180501\TempReplicaSynchronizationLog_ParksDemo_12DEV_SDE.txt
6: Deleted file F:\Parks\Logs\ParksDemo_Logs\PythonToolBox_Logs\NewLogs20180501\TempRepSchema_DBO.ParksDemo_Geo83_DataSrvr_v105_.xml
7: Deleted file F:\Parks\Logs\ParksDemo_Logs\PythonToolBox_Logs\NewLogs20180501\TempRepSchema_DBO.ParksDemo_WebMerc84_GISDEV_v105_.xml
8: Deleted file F:\Parks\Logs\ParksDemo_Logs\PythonToolBox_Logs\NewLogs20180501\TempRepSchema_DBO.ParksDemo_WebMerc84_GISPRO_v105_.xml
9: Deleted file F:\Parks\Logs\ParksDemo_Logs\PythonToolBox_Logs\NewLogs20180501\TempVersTablePostCompress.csv
10: Deleted file F:\Parks\Logs\ParksDemo_Logs\PythonToolBox_Logs\NewLogs20180501\TempVersTablePostCompress.txt.xml

Finally done.
```

Success Email

- To Any People Previously Connected/Booted
- Admin Email Address

```
# Use email list generated in first part
# of script to email users they may resume working
TO = emailList
SUBJECT = "Maintenance was Performed Successfully"
MSG = "Auto generated Message.\n\rServer maintenance w
exec(open(emailFilePy).read())
return
```

Message Intentionally Cut Off For Brevity

Error Email And Protocol

- If An Error Occurs
  - Log Error Details
  - Send Error Email
  - Try To Rebuild Versions

```
try:

    Run your code here

except Exception as e:
    masterLog.write("\n\r" + "Exception: " + str(e.message))
    TO = emailList
    SUBJECT = "Maintenance Encountered an Error"
    MSG = "Auto generated Message.\n\rServer maintenance encountered an error
    exec(open(emailFilePy).read())
    masterLog.write("\n\r" + "Exception raised, executing failsafe operation

                Message Intentionally Cut Off For Brevity

finally:

    Use this block to perform any cleanup, close documents,
    etc. This runs even if you have an exception
```

Questions?

- Email: michael.potts@tpwd.texas.gov