DateUnification.py

Disclaimer

This script is not supported under any CaseWare IDEA Inc. standard support program or service. The sample script is provided AS IS without warranty of any kind. CaseWare IDEA disclaims all implied warranties including, without limitation, any implied warranties of merchantability or of fitness for a particular purpose. The entire risk arising out of the use or performance of the sample script and documentation remains with you. In no event shall CaseWare IDEA, its authors, or anyone else involved in the creation, production, or delivery of the script be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss) arising out of the use of or inability to use the sample script or documentation, even if CaseWare IDEA has been advised of the possibility of such damages.

Purpose

The purpose of this script is to input an IDEA database, unify the date formats (in Date fields), and then import the database back into IDEA.

Requirements

You must have pywin32 installed. Pywin32 is automatically installed with IDEA versions that support Python.

Assumptions

- Numeric only dates come in the format of month-day-year
- Numeric only dates are either 6 characters (mmddyy) or 8 characters (mmddyyyy)
- Two-digit years with a value higher than 19 are assumed to come from the 1900s (i.e., 121287 would be parsed as 12/12/1987)
- Two-digit years with a value of 19 or less are assumed to come from the 2000s (i.e., 121212 would be parsed as 12/12/2012)
- If including the month name (i.e., Jan or January) the year is given in 4 characters (i.e., Jan 2nd 2012 would be valid but not jan 2nd 12)
- Files are assumed to be ASCII, and imported back as such
- Columns are counted starting at 1
- Dates are outputted as mm/dd/yyyy
- Dates are imported as Character fields

NOTE: Date Unification Sample Script is for testing purposes only

```
import win32com.client as win32ComClient # For communication with IDEA

import csv
import re
import tkinter as tk                # python 3
from tkinter import font  as tkfont # python 3
from tkinter import *
from tkinter import filedialog
from tkinter.messagebox import showinfo
import tempfile

###############################################
#                                             #
#   Two way Python/IDEA communication demo    #
#                                             #
###############################################

###############################################
#           Helper Functions                  #
def getFilename(path):
    f = list(path.split("\\"))[-1]
    return f
#                                             #
###############################################

###############################################
#           IDEA Comm Functions               #
def connectToIDEA():
    IDEA = None
    print("Attempting to connect to IDEA...")
    try:
        IDEA = win32ComClient.Dispatch(dispatch="Idea.IdeaClient")
        print("Connection to IDEA established")
    except:
        print ("Unexpected error:", sys.exc_info()[0])
        raise
    return IDEA

def exportDatabaseFromIDEA(IDEA,dbName,export,header):
    exportPath = export +"\\" + dbName+".del"
    # Begin the export
    db = IDEA.OpenDatabase(dbName)
    task = db.ExportDatabase()
    task.IncludeAllFields()
    if(header):
        task.IncludeFieldNames = "TRUE"
    else:
        task.IncludeFieldNames = "FALSE"
    eqn = ""
    task.Separators(",",".")
    task.PerformTask(exportPath,"Database","DEL",1,db.Count,eqn)
    print(dbName,"exported.")
```

```
def importDatabaseToIDEA(IDEA,dbName,export,header):
    #~~ Add error handling
    # Set up variables

    importPath = export+"\\"+dbName+".del"
    rdfPath    = export+"\\"+dbName+".rdf"
    dbName     = dbName+".IMD"
    #~~ check to see if file exists
    print("Importing",importPath,"into IDEA as",dbName)

    # Set up definition file
    defObj = IDEA.NewCSVDefinition()
    defObj.CsvFilePath = importPath
    defObj.DefinitionFilePath = rdfPath
    defObj.FieldDelimiter = ","
    defObj.TextEncapsulator = '"'

    if(bool(header)):
        defObj.FirstRowIsFieldNames = "TRUE"
    else:
        defObj.FirstRowIsFieldNames = "FALSE"
    defObj.CsvFileEncoding = 1

    IDEA.SaveCSVDefinitionFile(defObj)

    # Import into idea
    #~~ add option to do ascii / unicode?
    IDEA.ImportDelimFile(importPath, dbName, False, "", rdfPath ,
bool(header))
    IDEA.OpenDatabase(dbName)
    print(dbName,"imported.")
    return None

#                                              #
################################################

################################################
#          Delimited File Functions            #

def importDatabaseFromDEL(dbName,export):
    importPath = export +"\\"+dbName+".del"
    file = []
    with open(importPath,'r') as delFile:
        reader = csv.reader(delFile, delimiter = ",")
        for row in reader:
            file.append(row)
    file = list(map(list, zip(*file)))
    return file

def exportDatabaseToDEL(file,dbName,export):
    exportPath = export +"\\"+dbName+".del"
    quote = '"'
    delim = ","
    print("Exporting updated database to:",exportPath)
```

```
    with open(exportPath,'w') as delFile:
        for line in file:
            newLine = ""
            for item in line:
                if(delim in item):
                    item = quote+item+quote
                newLine = newLine + item + ","
            newLine = newLine[:-1] + "\n"
            delFile.write(newLine)
    return None
#                                                 #
###################################################

###################################################
#         File Manipulation Functions             #

def parseColsFromFile(file,cols,header):
    targetCols = []
    for col in cols:
        targetCols.append(file[col][header:])
    return targetCols

def rebuildFile(file,newCols,cols,header):
    newFile = []
    for i in range(len(newCols)):
        file[cols[i]][header:] = newCols[i]
    file = list(map(list, zip(*file)))
    return file
#                                                 #
###################################################

###################################################
#           Date Filter Functions                 #

## Assumes date is coming in as DD/MM/YYYY

def cleanDate(date):
    month   = date[0]
    day = date[1]
    year  = date[2]

    if(len(day) == 1):
        day = "0" + day

    if(month.isnumeric()):
        if(len(month) == 1):
            month = "0" + month
    else:
        months =
["","JANUARY","FEBRUARY","MARCH","APRIL","MAY","JUNE","JULY","AUGUST","SEPTEM
BER","OCTOBER","NOVEMBER","DECEMBER"]
        i = 0
        for curMonth in months:
            if(month.upper() in curMonth):
```

```
                month = i
                break
            i += 1
        if(len(str(month)) == 1):
            month = "0"+str(month)

    if(len(year) == 2):
        prefix = ""
        if(int(year) <= 19):
            prefix = "20"
        else:
            prefix = "19"
        year = prefix + year
    date = month+"/"+day+"/"+year

    return date

def parseDate(inputDate):
    fixedDate = ["","",""] # day,month year
    if(re.search('[a-zA-Z]', inputDate)): #check if it has alphabet
characters
        # has alphabet characters
        candidates=["","",""]
        i=0
        item = inputDate
        candidate = 0
        while(i < len(item)):
            while((i < len(item)) and re.search('[a-zA-Z0-9]',item[i])):
                candidates[candidate] += item[i]
                i+=1
            candidate += 1
            while((i < len(item)) and not re.search('[a-zA-Z0-9]',item[i]) ):
                i+=1
        #got our pieces
        for item in candidates:
            if item.isnumeric():
                #either date or year
                if(len(item) > 2):
                    fixedDate[2] = item # is year
                else:
                    fixedDate[1] = item # is date
            else:
                #not a number
                if(re.search('[0-9]',item)):
                   # has number therefor day
                   fixedDate[1] = re.sub('[^0-9]','', item)
                else:
                   # is month
                   fixedDate[0] = item
    else:
        #all numbers, maybe delimiters
        if(inputDate.isnumeric()):
            #no delimiters
            inputDate = str(inputDate)
            if(len(inputDate) == 6):
```

```
                fixedDate[0] = inputDate[:1]  # Month
                fixedDate[1] = inputDate[1:2] # Day
                fixedDate[2] = inputDate[2:]  # Year
            else:
                fixedDate[0] = inputDate[:2]  # Month
                fixedDate[1] = inputDate[2:4] # Day
                fixedDate[2] = inputDate[4:]  # Year
        else:
            fixedDate[0],fixedDate[1],fixedDate[2] = (re.sub('[^0-9]','|',
inputDate)).split("|")
            #there are delimiters
    return cleanDate(fixedDate)

def unifyDates(file,cols,header):
    print("Beginning to unify the dates")
    fixedCols = []
    colsToFix = parseColsFromFile(file,cols,header)
    for column in colsToFix:
        tempCol = []
        for value in column:
            tempCol.append(parseDate(value))
        fixedCols.append(tempCol)
    file = rebuildFile(file,fixedCols,cols,header)
    print("All dates have been unified")
    return file
#                                             #
###############################################

###############################################
#               UI controls                   #

def popup_showinfo(status,text):
    showinfo(status,text)

class appUI(tk.Tk):
    IDEA            = ""
    databaseName    = ""
    databaseData    = ""
    exportLocation  = ""
    header          = ""
    columns         = []
    extension       = ""
    newDatabaseName = ""
    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)

        self.title_font = tkfont.Font(family='Helvetica', size=18,
weight="bold")

        # the container is where we'll stack a bunch of frames
        # on top of each other, then the one we want visible
        # will be raised above the others
        self.container = tk.Frame(self)
        self.container.pack(side="top", fill="both", expand=True)
        self.container.grid_rowconfigure(0, weight=1)
```

```
        self.container.grid_columnconfigure(0, weight=1)

        self.exportLocation = tempfile.mkdtemp()

        self.frames = {}
        try:
            self.IDEA = connectToIDEA()
        except:
            popup_showinfo("Warning","Error connecting to IDEA")
            self.destroy()
            exit(1)
        for F in (ImportPage,ExportPage):
            page_name = F.__name__
            frame = F(parent=self.container, controller=self)
            self.frames[page_name] = frame

            # put all of the pages in the same location;
            # the one on the top of the stacking order
            # will be the one that is visible.
            frame.grid(row=0, column=0, sticky="nsew")
        self.show_frame("ExportPage")

    def __del__(self):
        shutil.rmtree(self.exportLocation)

    def show_frame(self, page_name):
        '''Show a frame for the given page name'''
        frame = self.frames[page_name]
        frame.tkraise()

    def import_data(self,header,file):
        self.header = bool(header.get())
        self.databaseName =file["text"]
        exportDatabaseFromIDEA(self.IDEA,self.databaseName,self.exportLocatio
n,self.header)
        self.databaseData =
importDatabaseFromDEL(self.databaseName,self.exportLocation)
        columnNames = []
        for i in range(len(self.databaseData)):
            if (self.header):
                columnNames.append(str(self.databaseData[i][0]))
            else:
                columnNames.append("Column "+str(i+1))
        page_name = ColumnsPage.__name__
        frame = ColumnsPage(parent=self.container, controller=self, cols =
columnNames)
        self.frames[page_name] = frame

        # put all of the pages in the same location;
        # the one on the top of the stacking order
        # will be the one that is visible.
        frame.grid(row=0, column=0, sticky="nsew")
        self.show_frame("ColumnsPage")

    def update_file(self,label,control):
```

```
        try:
            db = control.IDEA.CurrentDatabase()
        except:
            popup_showinfo("Warning","Error connecting to current database,
pleae ensure you have a database open")
            self.destroy()
            exit(1)
        dbname = getFilename(db.name)
        label.config(text=dbname)

    def columnsSelected(self,lstbox):
        cols = list(lstbox.curselection())
        if(len(cols) is 0):
            popup_showinfo("Warning","Please select the date columns you want
to unify")
        else:
            self.columns = cols
            self.show_frame("ImportPage")

    def importUpdatedDatabase(self,entry):
        newDB = entry.get()
        exclude = ["\\","/",":","*","?",'"',"<",">","[","]","|"]
        for e in exclude:
            if e in newDB:
                popup_showinfo("Warning","Invalid character '"+ e +"' in " +
newDB)
        try:
            self.databaseData =
unifyDates(self.databaseData,self.columns,self.header)
            exportDatabaseToDEL(self.databaseData,newDB,self.exportLocation)
            importDatabaseToIDEA(self.IDEA,newDB,self.exportLocation,self.hea
der)
            popup_showinfo("Success",newDB + " Successfully imported!")
            self.destroy()
        except e :
            self.show_frame("ExportPage")

class ExportPage(tk.Frame):

    def __init__(self,parent,controller):

        tk.Frame.__init__(self, parent)
        self.controller = controller
        try:
            db = controller.IDEA.CurrentDatabase()
        except:
            popup_showinfo("Warning","Error connecting to current database,
pleae ensure you have a database open")
            self.destroy()
            exit(1)
        dbname = getFilename(db.name)
        label = tk.Label(self, text="Current database",
font=controller.title_font)
        fileLabel = tk.Label(self, text=dbname, font=controller.title_font)
        label.pack(side="top", fill="x", pady=10)
```

```
        header = IntVar()

        button1 = tk.Button(self, text="Update",
                            command=lambda:
controller.update_file(fileLabel,self.controller))
        check = tk.Checkbutton(self, text="First Row as Field Headers",
variable=header)
        button2 = tk.Button(self, text="Continue",
                            command=lambda:
controller.import_data(header,fileLabel))

        fileLabel.pack()
        button1.pack()
        check.pack()
        button2.pack()

class ColumnsPage(tk.Frame):

    def __init__(self, parent, controller,cols):
        tk.Frame.__init__(self, parent)
        self.controller = controller
        label = tk.Label(self, text="Select Columns to Unify",
font=controller.title_font)
        self.grid(column=0, row=0)
        lstbox = Listbox(self, selectmode=MULTIPLE, width=20,
height=len(cols))
        for item in cols:
            lstbox.insert(END,item)
        label.pack(side="top", fill="x", pady=10)
        # attach listbox to scrollbar
        scrollbar = tk.Scrollbar(self)
        scrollbar.pack(side=RIGHT, fill=Y)
        lstbox.config(yscrollcommand=scrollbar.set)
        scrollbar.config(command=lstbox.yview)
        button = tk.Button(self, text="Select",
                           command=lambda:
controller.columnsSelected(lstbox))
        lstbox.pack()
        button.pack()

class ImportPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller
        label = tk.Label(self, text="Import Updated Database",
font=controller.title_font)
        label.pack(side="top", fill="x", pady=10)
        label = tk.Label(self, text="Enter the name of the new database")

        e1 = Entry(self)
        label.pack()

        e1.pack()
        button = tk.Button(self, text="Create new database",
```

```
                              command=lambda:
controller.importUpdatedDatabase(e1))
        button.pack()

#                                                 #
###################################################

###################################################
#           Main control Functions                #

# Entry point
if __name__ == "__main__":
    try:
        app = appUI()
        app.title("Date Unification")
        app.mainloop()
    except Exception as e:
        popup_showinfo("Warning","Error running program")
        print(e)
#                                                 #
###################################################
```