```
#ExportTableWithDomainDescriptions

'''
Author:
  Kurt Sargent
  Barnes and Duncan
  Winnipeg, Manitoba, Canada
  kurtsargent@gmail.com

Date created March 23 2011
Modified

Purpose:
  Using several python dictionaries, will iterate through a feature class
  table and create a new table. Any fields with coded value domains
  will have the descriptions written to the new table instead of codes.
  Script will skip objectid, shape, shape_Length, shape_area fields. More
  fields can easily be added to the "SkipList"

Properties (right-click on the tool and specify the following)
General
  Name   ExportTableWithDomainDescriptions
  Label  Export Table With Domain Descriptions
  Desc   Rewrites a feature class table, substituting domain descriptions
         for coded values where applicatble.
Source script ExportTableWithDomainDescriptions.py

Parameter list
                            Parameter Properties
          Display Name         Data type        Type      Direction  MultiValue
  argv[1]  Input feature class  Feature Layer    Required  Input      No

'''



# Import system modules
import traceback, sys, string, os, math, time, arcgisscripting

# Create the Geoprocessor object
gp = arcgisscripting.create(9.3)
gp.OverWriteOutput = 1


# Print Ascii Python just because I like it.
gp.addmessage("   ....... M..... .M ...  ................................")
gp.addmessage("   ....... .MMM8,MMMM,    ................................")
gp.addmessage("...      +M    .7    .M.  .    .  .. . .. .  ...   .  .  ...")
gp.addmessage("...     M.  IIM7M7N.   M  .    . . . . ..    .       . . .")
gp.addmessage("...     N   ?IMMOM7.   M .        .     .          ." )
gp.addmessage("...    .D    8M$MM.  ,M. .             .")
gp.addmessage("... MM M$MMM$$$$$8$$$$$OMM7.      .     .          ." )
gp.addmessage("...  MM$$$$7$$$$$$7$$$$7M ..      .               ." )
gp.addmessage("...   7$$$$$$$$$$$$$$$$I.... .   ..  . ..  ....   .  ."  )
gp.addmessage("   ..I$$$$$$$$$$$$$$$$$.                          .  ." )
gp.addmessage("   ...$$$$88M$$8NZ$$$$$.         .     .  .   . . ."  )
gp.addmessage("......Z$$$$OO$$88N$$$$$M .  ..... .. ......  .... ....   ..." )
gp.addmessage(" ....  $$$$$$$$$$$$$$M,          ." )
gp.addmessage("..... M.M$$$$$$$$$$ZM            .               ."  )
gp.addmessage("   .. M~?I$MD$77$DMM8M            .               ."  )
gp.addmessage("   ... $N....M8$$$$$.M" )
gp.addmessage(" .... MM .  .M8$$$MDN.            .               ." )
gp.addmessage("   ...7M . ..M$$7DM8M")
gp.addmessage(".....MMO....  =$$7DO88            .               ."  )
gp.addmessage(".....M.D. ..M7$$$ODN              .               ."  )
gp.addmessage("..... .=  .8?$$$$$$M              .               8      =" )
gp.addmessage("..... .....N$$$$NDDZ              .               ,      ."  )
gp.addmessage("..........NZ$$$MN8N               .               .D. M ,"  )
gp.addmessage("..... ..  MI$$$MDOM            ..MMMM.           .   M"  )
gp.addmessage("..... .. 7=Z$$$$MNM          MO$$D888MMM             M"  )
gp.addmessage("...    . M=$$$$$$$M.       M8NN$$MDDDN$$$M.    .    .  $M"  )
gp.addmessage("...    . 7=7$$$$$8DM.   MM888MM$$$$$$$$$D8M    .    .  $M." )
gp.addmessage("...    .  $7$$$$$DD88NM$$MDDD7$$$$$$$$$$DM88M ..    ..$M"  )
gp.addmessage("  .    .  M=$$$$$8DD8NM$$$$$$$$ZZ==Z=O$$$DMM$$MI  7M$8M"  )
gp.addmessage("...    . . M~$$$$$$7$7$$$$$$$$=MM . .7MZ$$$$$$NO88DMZM."  )
gp.addmessage("... . .. .  M=M$$$$$$$$$$$$O=M.    .    ,M=$$$$$MNNM8M" )
gp.addmessage("..... .. .  ..M=ID$$$$$$$Z$M..    .     ..M7$$$$$$?O" )
gp.addmessage("..... .. .  ..~$MMMMMMMM$..       .        .MMM$MM.." )
gp.addmessage("...    .         I8MMMMMMI.       .      ..  IMMMMM7." )
```

```
gp.addmessage("\n")
gp.addmessage("\n")
gp.addmessage("\n")
gp.addwarning("############################################################")
gp.addwarning("#               Exports a feature class to a               #")
gp.addwarning("#              geodatabase table and where fields          #")
gp.addwarning("#               have coded value domains, the              #")
gp.addwarning("#               descriptions are written to the            #")
gp.addwarning("#                       new table                          #")
gp.addwarning("#                     Kurt Sargent                         #")
gp.addwarning("#                   Barnes and Duncan                      #")
gp.addwarning("#                    March 23, 2011                        #")
gp.addwarning("#                  kurtsargent@gmail.com                   #")
gp.addwarning("#                                                          #")
gp.addwarning("#            Last Updated March 23, 2011                   #")
gp.addwarning("############################################################")
gp.addmessage("\n")

gp.AddToolbox("C:/Program Files (x86)/ArcGIS/ArcToolbox/Toolboxes/Data Management Tools.tbx")
gp.AddToolbox("C:/Program Files (x86)/ArcGIS/ArcToolbox/Toolboxes/Analysis Tools.tbx")

# Establish parameters for the domainToTable tool that will be run later
codeFld= "CODE"
descptnFld = "DESCRIPTION"
domainFld = "DomainName"

try:

    # Input FC from Form
    Input_FC = gp.GetParameterAsText(0)
    Input_FC_Name = os.path.basename(Input_FC)
    if not gp.Exists(Input_FC):
        raise "Input_FC doesn't exist"
    else:
        gp.addmessage("\n"+"got input FC..." + "\n    " + Input_FC + "\n")

    # Get path to GDB where Input_FC resides and set that as
    # the current workspace.
    gdb = (os.path.split(Input_FC))[0]
    gp.WorkSpace = gdb
    gp.AddWarning("set work space to:  " + "\n    " + gdb + "\n")


    # Create a new table. This is where the Input_FC's values will
    # be copied to, with domain descriptions replaceing coded values.
    # The new table name will be the Input_FC's name with '_domainDesc'
    # added to it.
    Input_FC_Name = os.path.basename(Input_FC)
    result = gp.CreateTable_management (gdb, Input_FC_Name + "_domainDesc")
    outputDomainTable = result.GetOutput(0)
    gp.AddMessage("Created " + Input_FC_Name +
        "_domainDesc table" + "\n" + outputDomainTable)
    del result


    # domainDesc table needs fields from the Input_FC. However, some need
    # to be skipped, such as objectID, shape, etc. Any other fields can be
    # added to the skip list here. Alternatively, a list of skip fields
    # could be derived from user input on the form in a field map.
    #

    # describe Input_FC for list of skip fields
    dsc = gp.Describe(Input_FC)
    skipList = [dsc.oidfieldname, dsc.shapefieldname,
                dsc.shapefieldname + "_Length",
                dsc.shapefieldname + "_Area", "REVIEW" ]

    # Basically, we are going to copy the fields from Input_FC over to
    # the new domainDesc table. However, if a field has a coded value
    # domain associated with it, we need to make that field a TEXT field
    # to hold the description

    # make list of all domains in the inputFC so when we run domainToTable
    # later it will only do so for domains associated with the inputFC
    inputFCsDomains = []
    Fields = gp.ListFields(Input_FC)
    for aField in Fields:
        # get field name and test if in skipList
        field_name = aField.name
        if field_name not in skipList:
            aDomain = aField.domain
            # if field does NOT have a domain, we want to copy the attibutes.
            # however, some 'get' properties are not consitent with the
```

```
            # respective 'set' properties. E.g. field type from a text field
            # returns as 'String', but to create a string field, the property
            # must be 'TEXT'. There may be other property issues that arise
            # from various other field types and they can be added here.
            if aDomain == "":
                field_type = aField.type
                if field_type == "String":
                    field_type = "TEXT"
                elif field_type == "Integer" or field_type == "SmallInteger":
                    field_type = "LONG"
                field_precision = aField.Precision
                field_scale = aField.scale
                field_length = aField.length
                # add a field based on the properties set above.
                gp.AddField_management(outputDomainTable, field_name,
                    field_type, field_precision, field_scale, field_length)
            else:
                # if a domain is associated with the field we make it a
                # standard text field. I chose 50 as the field width but
                # it can be changed to accomodate larger domain descriptions.
                inputFCsDomains.append(aDomain)
                gp.AddField_management(outputDomainTable, field_name,
                     "TEXT", "", "", 50)
    del aField, Fields

    # Use Describe method to return geodatabase domains
    descGDB = gp.Describe(gdb)
    gdbDomains = descGDB.Domains
    gdbDomains.sort()


    # Loop through each domain and excecute DomainToTable_management,
    # add a field called DomainName and populate that field with the
    # respective domain name.
    #

    '''
    ****Dictionaries Explained******
    A series of dictionaries is used to create a list of dictionaries containing
    filed values to populate to the new domainDesc table. First is the
    domainNameDict. The domainDict stores the domain(name) as the key, and the
    codeDescriptionDict as its object. The codeDescriptionDict stores each
    doamins coded value as its key, and the respective description as its
    object. So, essentially, the domainNameDict will allow us to use each
    Domain name to  call the appropriate codeDescriptionDict and pass it a
    coded value and return the correct description for each field for each
    row of the Input_FC.

    As we iterate through the Input_FC, and iterate through all the field
    we will populate fldNameValueDict, which stores the fieldName as its key
    and field value (field value will be the existing field value for a non
    domain field, and the coded value's description of a domain field). So,
    fldNameValueDict is created for each row of the Input_FC and added to the
    insertRowList. Later, for each fldNameValueDict in insertRowList, we'll
    set each field value based on the fldNameValueDict value returned by passing
    it the field name. Once all field values are set, we will insert that row
    into the table.
    '''
    # domainNameDict{domain, codeDescriptionDict}
    domainNameDict = {}
    for domain in inputFCsDomains:

        # codeDescriptionDict{codedValue, Description}
        codeDescriptionDict = {}
        # Make domainTableName. We are using the same gdb as
        # the Input_FC but can change to scratch workspace or whatever.
        domainTableName = gdb + "\\" + domain

        # create table of domain descriptions and codes
        result= gp.DomainToTable_management (gdb, domain,
                                             domainTableName,
                                             codeFld, descptnFld)
        domainTable = result.GetOutput(0)
        del result

        # Add DomainName field to domainTable

        gp.AddMessage("created domain table: " + domain + "\n")
        # use update cursor to "calcualte" the domainFld with the
        # respective domain name for use in dictonary later
        rows = gp.SearchCursor(domainTable)
        for row in iter(rows.next, None):
```

```
            # get code value
            aCode = row.GetValue(codeFld)
            # get description value
            aDescription = row.GetValue(descptnFld)
            # add to codeDescriptionDict(aCod, aDescription)
            codeDescriptionDict[aCode] = aDescription

        # add completed codeDescriptionDict to the domainNameDict
        domainNameDict[domain] = codeDescriptionDict
        del row, rows,codeDescriptionDict

    # Iterate through Input_FC
    insertRowList = []
    # Create search cursor for Input_FC iteration
    rows = gp.SearchCursor(Input_FC, "", "", "", "")
    # Field list for Input_FC
    fieldList = gp.ListFields(Input_FC)
    gp.AddMessage("Iterating through Input_FC to build dictionaries" + "\n")

    for row in iter(rows.next, None):
        fldNameValueDict = {}
        for aField in fieldList:
            # field value list to contain fileName (key) and
            # aFieldVal (obj)
            aFieldName = aField.Name
            #gp.AddMessage(aFieldName)
            if aFieldName not in skipList:
                aFieldVal = row.GetValue(aFieldName)
                aFieldDomain = aField.Domain
                # if no domain associated with the field then
                # accept the current value
                if aFieldDomain == "":
                    # set fldNameValueDict key(fieldName)/object(fieldValue)
                    fldNameValueDict[aFieldName] = aFieldVal
                else:
                    # retrive appropriate domian codeDescriptionDict
                    # from the domainNameDict
                    codeDict = domainNameDict[aFieldDomain]
                    # retrieve appropriate domain description
                    codeDesc = codeDict[aFieldVal]
                    # set fldNameValueDict key(fieldName)/object(fieldValue)
                    fldNameValueDict[aFieldName] = codeDesc

        # add completed fldNameValueDict for current row to insertRowList
        insertRowList.append(fldNameValueDict)
    del row, rows, aFieldVal,aFieldDomain, aFieldName,fldNameValueDict

    # Create insertCursor to add new records to the export table
    rows = gp.insertCursor(outputDomainTable)
    fieldList = gp.ListFields(outputDomainTable)
    gp.AddMessage("\n" + "\n" + "populating domainDesc table")

    rowCount = 0
    # each fldNmValDict contains the values for all fields in the current row.
    for fldNmValDict in insertRowList:
        row = rows.NewRow()
        #Set values for all field not in skipList
        for aField in fieldList:
            aFieldName = aField.name
            if aFieldName not in skipList:
                aValue = fldNmValDict[aFieldName]
                row.SetValue(aFieldName, aValue)
                #gp.AddMessage("Set: " + aFieldName + " to: " + str(aValue))
        rows.InsertRow(row)

    # ***** Clean up *****
    #
    #get rid of domain tables
    for domain in inputFCsDomains:
        domainTableName = gdb + "\\" + domain
        if gp.Exists(domainTableName):
            try:
                gp.Delete_management (domainTableName)
                gp.AddMessage("deleted domain table " + domain )
            except:
                gp.AddError("Failed to delete " + domainTableName)
    gp.AddMessage("\n")

    # Compact database
    gp.Compact_management (gdb)
    gp.AddMessage("Compacted GDB")

    del gdbDomains, domain, fieldList, aField, gdb,
```

```
    del outputDomainTable, Input_FC,domainNameDict, insertRowList, rows, row
    del gp



except "domainTable(s) already exist...please delete":
    gp.AddError("domainTable(s) already exist...please delete")
except "Input_FC doesn't exist":
    gp.AddError("Input_FC does not exist")
except arcgisscripting.ExecuteError:
    # Get the geoprocessing error messages
    #
    msgs = gp.GetMessage(0)
    msgs += gp.GetMessages(2)

    # Return gp error messages for use with a script tool
    #
    #gp.AddReturnMessage(msgs)
    gp.AddError(msgs)

    # Print gp error messages for use in Python/PythonWin
    #
    #print msgs
except:
    # Get the traceback object
    #
    tb = sys.exc_info()[2]
    tbinfo = traceback.format_tb(tb)[0]

    # Concatenate information together concerning the error into a
    #   message string
    #
    pymsg = tbinfo + "\n" + str(sys.exc_type)+ ": " + str(sys.exc_value)

    # Return python error messages for use with a script tool
    #
    gp.AddError(pymsg)

    # Print Python error messages for use in Python/PythonWin
    #
    print pymsg
```