Use Field Mapping and Python Scripting to Make Your Job Easier

By Drew Flater, Product Engineer, Esri Geoprocessing Team

A number of geoprocessing tools, including Spatial Join (Analysis), Append (Management), Merge (Management), Feature Class To Feature Class (Conversion), and Table To Table (Conversion), have a parameter for controlling how fields from input datasets are processed and written or mapped to the output dataset—the Field Map parameter. In addition to the simple moving of attributes from input to output, field mapping can also be useful for some common tasks such as field concatenation and calculating statistics like mean, sum, and standard deviation.

If you haven't used the Field Map parameter before, you should. Understanding and using field mapping will often reduce the number of processing steps in a workflow and ensure that, in any scenario, attributes are handled in an appropriate way. Yes, the Field Map parameter is a complicated one, but it is well worth the time it takes to figure it out.

Because it is a complicated parameter, working with it in Python can also be complicated. The best way to interact with field mapping in Python scripting is with the FieldMappings object.

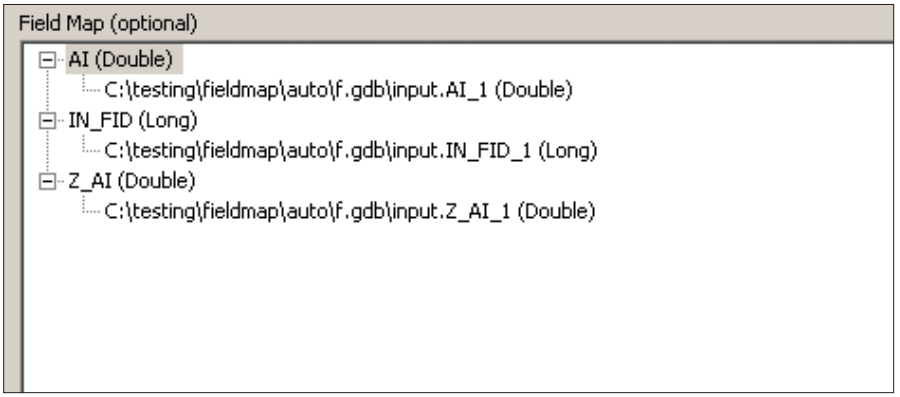
In Python, most geoprocessing tool parameter types are seen as simple numbers or strings (specifying a feature class input is as easy as providing the path to the feature class). But several of the more complex parameters have objects that exist to help you effectively work with the parameter. The Field Map parameter can accept a long structured string indicating the field map settings (you may have seen this long string equivalent in geoprocessing messages); however, working with the field mapping string is inefficient and error prone, so use the FieldMappings object for the best experience.

Problem

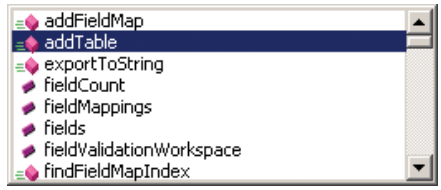
I was recently presented with a data migration problem where field mappings and Python scripting literally saved me weeks of work. The goal was to convert a collection of hundreds of Vector Product Format (VPF) databases containing many feature classes to a handful of geodatabases, and because of the large scale of the migration, it had to be accomplished in an automated fashion (thus saving many weeks of work). The schema of the geodatabases was already set up with a number of feature datasets and empty feature classes into which the VPF feature class data would be imported using the Append (Management) tool.

The iteration through the collection of VPF databases was solved with some simple looping techniques involving the arcpy.ListDatasets() and arcpy.ListFeatureClasses() functions. However, there was a fundamental problem that nearly derailed the automation of this process: VPF feature classes can have spaces in their field names, while geodatabase datasets cannot. When the empty geodatabase feature classes were created from the schema of the VPF feature classes, the spaces in the field names were automatically changed to underscores (_) in the geodatabase feature classes. This very subtle difference caused huge ripples in the automated process because the Append tool cannot automatically match fields like mcc description to mcc_description. In the

↓ Append (Management) tool default Field Map, showing the target geodatabase feature class schema



```
>>> fieldmappings = arcpy.FieldMappings()
>>> fieldmappings.
```



↑ The FieldMappings object has many properties and methods to efficiently work with field mapping.

output geodatabase feature class, all the values in the mcc_description field are NULL because the fields were not matched.

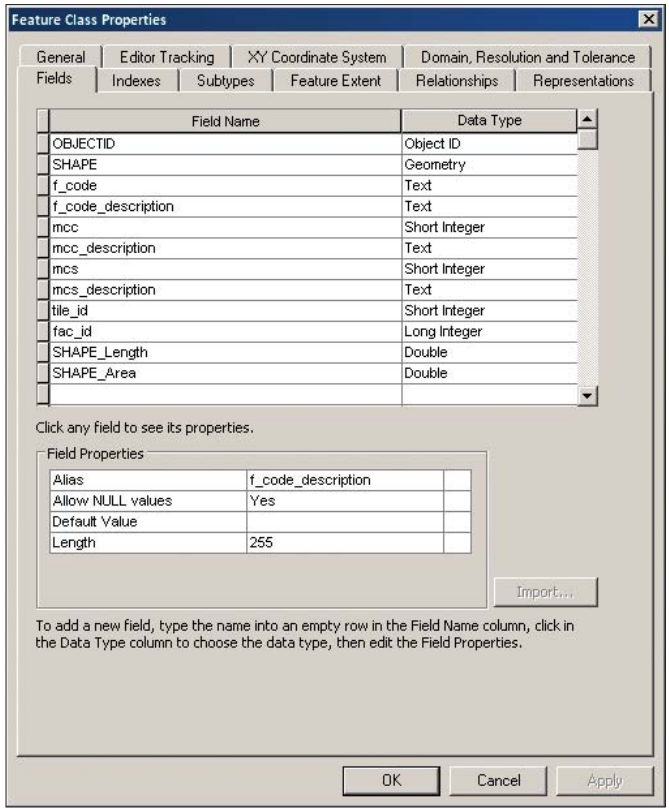
Viewing the Field Map parameter on the Append tool helps illustrate the issue. The fields f_code description, mcc description, and mcs description are not automatically matched to the geodatabase target schema because of the difference of the space and underscore characters.

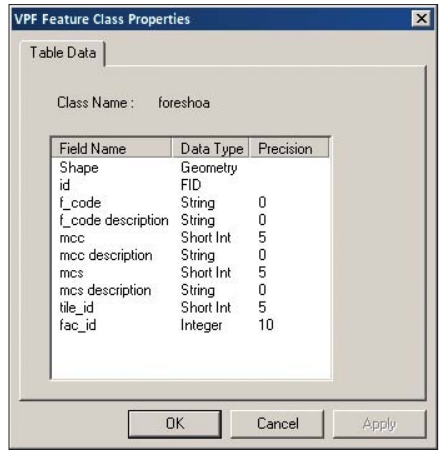
Solution

This situation can be rectified by manually adding the matching input fields using the Field Map controls on the Append tool. On each field that does not have a match, right-click and add the appropriate input field from the VPF feature class.

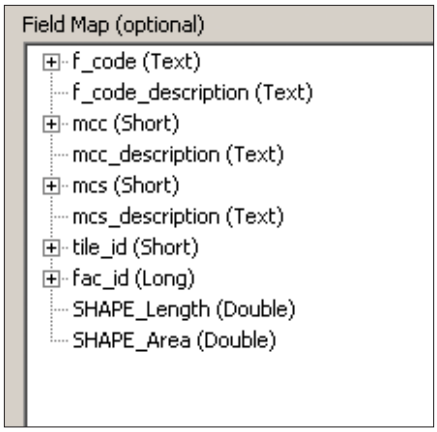
These manual steps resulted in a correct field mapping and import of the VPF feature class data to the geodatabase for a single VPF feature class. To successfully perform the automated migration, I incorporated this additional field mapping work into the migration script using the FieldMappings object. For each iteration before a VPF feature class is appended into a geodatabase feature class, the script iterates through each VPF feature class field, does a find and replace for those fields with a space and the corresponding field with an underscore, and matches the two fields in the FieldMappings object.

A good way to understand the usage of the FieldMappings object is to think about these steps in the same context as when field mapping is done in the tool dialog. The script goes through each output field in the Field Map (these are the fields from the target dataset) and adds an input field from the VPF feature class that matches, regardless of the differences in space and underscore characters, before finally executing the Append process with the customized field mapping.





↑ VPF and geodatabase feature class fields



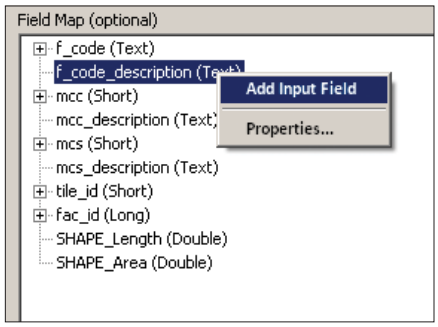
↑ The Append tool default Field Map, showing the target geodatabase feature class schema

Summary

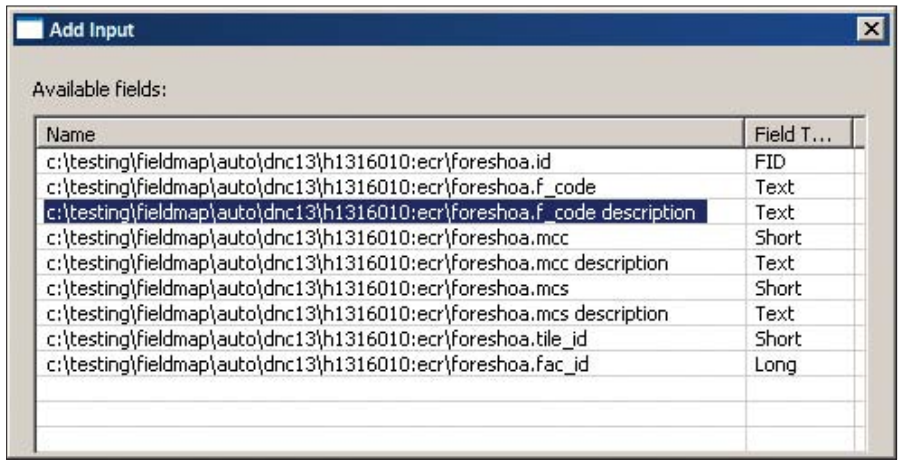
Several geoprocessing tools use the complex but powerful Field Map parameter to control how attributes from input datasets are mapped to the output dataset. Understanding and using the Field Map parameter on geoprocessing tools can enable time-saving workflows; will often reduce the number of required processing steps; and can ensure that, in any scenario, attributes are never lost and always handled in an appropriate way. In Python scripting, the FieldMappings object provides efficient, automatable access to all the same functionality available with the Field Map control on the tool dialog and can be used in many scenarios including programmatically matching fields with different names. Use field mapping—it will make your job easier.

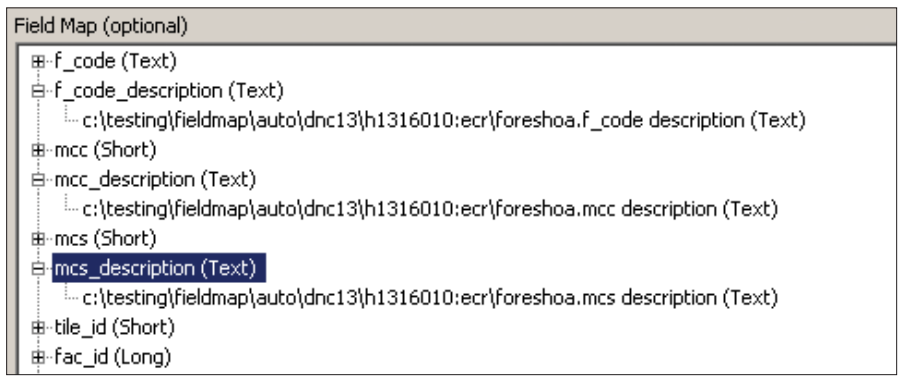
About the Author

Drew Flater has been designing, testing, and writing about geoprocessing tools and spatial analysis for Esri since 2008. He loves creating models and scripts that solve spatial problems and make GIS analysis and data management tasks easier.



↑ Use the Field Map controls to do manual field mapping





```
import arcpy
import os

folder = r"C:\testing\fieldmap\auto\dnc13"
gdbfolder = r"C:\testing\fieldmap\auto\gdbs"

arcpy.env.workspace = folder
listvpf = arcpy.ListDatasets()

for vpf in listvpf:
    # First character of the VPF determines the geodatabase to append
to
    #
    if vpf[0] == "h":
        gdb = os.path.join(gdbfolder, "DNCHarbor.gdb")
    elif vpf[0] == "a":
        gdb = os.path.join(gdbfolder, "DNCApproach.gdb")
    elif vpf[0] == "c":
        gdb = os.path.join(gdbfolder, "DNCCoastal.gdb")
    elif vpf[0] == "g":
        gdb = os.path.join(gdbfolder, "DNCGeneral.gdb")

    # The characters after the : in the VPF name determine the fea-
ture dataset to append to
    #
    fd = vpf.split(":")[1]
    arcpy.env.workspace = os.path.join(folder,vpf)
    listvpffc = arcpy.ListFeatureClasses()
    for fc in listvpffc:
        targetd = os.path.join(gdbfolder,gdb,fd,fc)

        # Create FieldMappings object and load the target dataset
        #
        fieldmappings = arcpy.FieldMappings()
        fieldmappings.addTable(targetd)

        # Loop through each field in the input dataset
        #
        inputfields = [field.name for field in arcpy.ListFields(fc) if
not field.required]
        for inputfield in inputfields:
            # Iterate through each FieldMap in the FieldMappings
            #
            for i in range(fieldmappings.fieldCount):
                fieldmap = fieldmappings.getFieldMap(i)
                # If the field name from the target dataset matches
to a validated input field name
                #
                if fieldmap.getInputFieldName(0) == inputfield.replace("
", "_"):
                    # Add the input field to the FieldMap and replace
the old FieldMap with the new
                    #
                    fieldmap.addInputField(fc, inputfield)
                    fieldmappings.replaceFieldMap(i, fieldmap)
                    break

        # Perform the Append
        #
        arcpy.management.Append(fc, targetd, "NO_TEST", fieldmappings)
```