Working with the ArcPy Data Access Module

Dave Wynne, Jason Pardy
Abstract

In this workshop, see highlights and demonstrations of the new data access module (arcpy.da) for accessing and working with data.

Major topics to be covered include starting editing sessions and operations, fast cursor support, converting data to and from NumPy arrays, Walk function for walking data types in folder or geodatabase, and managing geodatabase objects (domains, subtypes, versions, and replicas).

Please fill out your session surveys
Session ID: 64
A new module (at 10.1) for working with data

<table>
<thead>
<tr>
<th>arcpy.da</th>
<th>Faster cursors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Edit session support</td>
</tr>
<tr>
<td></td>
<td>NumPy array conversion</td>
</tr>
<tr>
<td></td>
<td>Replicas, versions, subtypes, and domains</td>
</tr>
<tr>
<td></td>
<td>Walk</td>
</tr>
</tbody>
</table>
Cursors

• Cursors provide record-by-record access
  - Basic necessity for many workflows

<table>
<thead>
<tr>
<th>arcpy.da.SearchCursor</th>
<th>Read-only access</th>
</tr>
</thead>
<tbody>
<tr>
<td>arcpy.da.UpdateCursor</td>
<td>Update or delete rows</td>
</tr>
<tr>
<td>arcpy.da.InsertCursor</td>
<td>Insert rows</td>
</tr>
</tbody>
</table>

• The ‘classic’ cursor model works, but…
  - Not fast enough
  - Bottleneck for some Python workflows
Cursors

- `SearchCursor(in_table, field_names, where_clause=None, spatial_reference=None, explode_to_points=False, sql_clause=(None, None))`

- `UpdateCursor(in_table, field_names, where_clause=None, spatial_reference=None, explode_to_points=False, sqlClause=(None, None))`

- `InsertCursor(in_table, field_names)`
Cursors

- arcpy.da cursors use lists and tuples
  - Row values are accessed by index
    
    ```python
    fields = ['field1', 'field2']
cursor = arcpy.da.InsertCursor(table, fields)
cursor.insertRow([1, 10])
    ```

- Different from ‘classic’ cursors
  - Work with row objects
  - Row values are accessed with setValue/getValue properties

    ```python
    cursor = arcpy.InsertCursor(table)
    row = cursor.newRow()
    row.setValue("field1", 1)
    row.setValue("field2", 10)
    cursor.insertRow(row)
    ```
with statements

• arcpy.da Cursors support with statements

```python
with arcpy.da.SearchCursor(table, field) as cursor:
    for row in cursor:
        print row[0]
```

• **With** statement
  - Provide clarity
  - Other benefits: such as allowing edits on multiple tables in the same workspace
Fields and tokens

- Use only those fields you need

- Tokens can be also be used
  - Get only what you need: asking for full geometry is expensive

<table>
<thead>
<tr>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID@</td>
</tr>
<tr>
<td>SHAPE@XY</td>
</tr>
<tr>
<td>SHAPE@TRUECENTROID</td>
</tr>
<tr>
<td>SHAPE@X</td>
</tr>
<tr>
<td>SHAPE@Y</td>
</tr>
<tr>
<td>SHAPE@Z</td>
</tr>
<tr>
<td>SHAPE@M</td>
</tr>
<tr>
<td>SHAPE@</td>
</tr>
<tr>
<td>SHAPE@AREA</td>
</tr>
<tr>
<td>SHAPE@LENGTH</td>
</tr>
<tr>
<td>SHAPE@JSON     *</td>
</tr>
<tr>
<td>SHAPE@WKB      *</td>
</tr>
<tr>
<td>SHAPE@WKT      *</td>
</tr>
<tr>
<td>* - added at 10.1 sp1</td>
</tr>
</tbody>
</table>
Cursors
Editor class

- Supports edit sessions and edit operations
- Edits are temporary until saved and permanently applied
- Can quit an edit session without saving changes
- When?
  - To edit feature classes that participate in a...
    - Topology
    - Geometric network
  - Versioned datasets in ArcSDE geodatabases
  - Some objects and feature classes with class extensions
Editor / with statements

- Editor also supports `with` statements
  - Handle appropriate start, stop and abort calls for you

```python
with arcpy.da.Editor(workspace) as edit:
    # your edits
```

Open an edit session and start an edit operation

Exception—operation is aborted, and edit session is closed without saving

No exceptions—stop the operation and save and close the edit session
Editor class

<table>
<thead>
<tr>
<th>Editor methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startEditing({with_undo}, {multiuser_mode})</td>
<td>Starts an edit session.</td>
</tr>
<tr>
<td>stopEditing(save_changes)</td>
<td>Stops an edit session.</td>
</tr>
<tr>
<td>startOperation()</td>
<td>Starts an edit operation.</td>
</tr>
<tr>
<td>stopOperation()</td>
<td>Stops an edit operation.</td>
</tr>
<tr>
<td>abortOperation()</td>
<td>Aborts an edit operation.</td>
</tr>
<tr>
<td>undoOperation()</td>
<td>Undo an edit operation (roll back modifications).</td>
</tr>
<tr>
<td>redoOperation()</td>
<td>Redoes an edit operation.</td>
</tr>
</tbody>
</table>

- If you’ve worked with edit sessions in ArcObjects, this will look familiar
More list functions (and objects) to support workflows

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListDomains</td>
<td>Lists the attribute domains belonging to a geodatabase.</td>
</tr>
<tr>
<td>ListReplicas</td>
<td>Lists the replicas in a workspace.</td>
</tr>
<tr>
<td>ListSubtypes</td>
<td>Return a dictionary of the subtypes for a table or feature class.</td>
</tr>
<tr>
<td>ListVersions</td>
<td>List the versions in a workspace.</td>
</tr>
</tbody>
</table>
Demo

arcpy.da
List functions
NumPy support

- NumPy is a 3rd party library for scientific computing
- arcpy.da supports converting tables and feature classes to/from NumPy

- RasterToNumPyArray & NumPyArrayToRaster
  - Added at 10.0 to support converting rasters to and from numpy arrays
NumPy functions

- At 10.1, `arcpy.da` provides additional support for tables and feature classes

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FeatureClassToNumPyArray</td>
<td>Convert a feature class to an array</td>
</tr>
<tr>
<td>TableToNumPyArray</td>
<td>Convert a table to an array</td>
</tr>
<tr>
<td>NumPyArrayToFeatureClass</td>
<td>Convert an array to a Feature Class</td>
</tr>
<tr>
<td>NumPyArrayToTable</td>
<td>Convert an array to a Table</td>
</tr>
<tr>
<td>ExtendTable</td>
<td>Join an array to a Table</td>
</tr>
</tbody>
</table>
Export to NumPy

- Can convert tables and feature classes into numpy arrays for further analysis

```python
import arcpy
import numpy

in_fc = "c:/data/usa.gdb/USA/counties"
field1 = "INCOME"
field2 = "EDUCATION"

array1 = arcpy.da.FeatureClassToNumPyArray(in_fc, [field1, field2])

# Print correlation coefficients for comparison of 2 fields
print numpy.corrcoef((array1[field1], array1[field2]))
```
Import from NumPy

• Take the product of your work in numpy and export it back to ArcGIS

```python
array1 = numpy.array([(1, (471316.3, 5000448.7)),
                      (2, (470402.4, 5000049.2))],
                     numpy.dtype([('idfield', numpy.int32),
                                  ('XY', '<f8', 2)]))

SR = arcpy.Describe(template).spatialReference

# Export the numpy array to a feature class using the XY # field to represent the output point feature
arcpy.da.NumPyArrayToFeatureClass(array1, outFC, ['XY'], SR)
```

• Need to output polygons, lines, multipoints?
  - http://esriurl.com/5862
Exporting to NumPy
Walk
*Added at 10.1 Service Pack 1

- `arcpy.da.Walk(top, topdown=True, onerror=None, followlinks=False, datatype=None, type=None)`

- Step through a directory structure to find ArcGIS data types
- Returns a tuple of three: path, path names, and filenames
- Same pattern as `os.walk`
- Prior to `arcpy.da.Walk`:
  - Cobble together various `arcpy List` functions with Python’s `os.walk` to navigate a directory structure
Walk

walk = arcpy.da.Walk(workspace, datatype=datatypes)
for path, path_names, data_names in walk:
    for data_name in data_names:
        do_something(os.path.join(path, data_name))

• Comparison:
  - Walk: http://esriurl.com/5931
  - The hard way: http://esriurl.com/5932
Walk
Thank you!

- ArcGIS Python resource center
Questions?

Please fill out your session surveys

Session ID: 64

@arcpy
esri
Understanding our world.