Creating Mosaic Datasets and Publishing Image Services using Python

Jie Zhang, Jamie Drisdelle

Session Offering ID: 305
Overview

• Introduction to mosaic dataset
• Raster product for sensor imagery
• Automatic mosaic dataset authoring workflow with python
  - To discovery raster data
  - To create mosaic dataset
  - To configure mosaic dataset
• Introduction to image service
• Publishing with python
  - To publish mosaic dataset as image service
  - To consume image service
What is the mosaic dataset?

- A geodatabase data model used to catalog and process your collections of imagery
  - Stored as a table and viewed as a table or image
- Indirect pixel management
  - Images can remain in their native format on disk or be loaded into the geodatabase
- Unlimited size*
- Provides dynamic mosaicking and the-fly processing
- License requirement – Standard or Advanced
Building a mosaic dataset

- Store in a geodatabase
  - Build with geoprocessing tools
  - Automation with models or Python
- Simple workflow
  1. Create mosaic dataset
  2. Add imagery (*raster type*)
  3. Optionally, edit properties and functions
- Can interactively edit and view in ArcMap
  - All layers are displayed
  - Edit and add fields in table window
Raster products

- Simplifies using sensor data
  - Quick and easy visualization of common band combinations
  - Simple drag-n-drop, less clicking
- Key metadata
  - Sensor name
  - Acquisition date
  - Wavelength
- Function templates
  - Multispectral, Pansharpen
- Temporary function raster dataset
Creating mosaic datasets with geoprocessing

- Mosaic Dataset toolset
  - Creation
    - Create Mosaic Dataset
    - Add Rasters To Mosaic Dataset …
  - Modify
    - Define Mosaic Dataset Nodata
    - Build Footprints …
  - Enhancement
    - Build Seamlines
    - Color Balance Mosaic Dataset …

- All tools are accessible through arcpy
Creating Mosaic Datasets with Python

Jie Zhang
Raster data management methods in arcpy

- Where are the APIs supports raster data?
  - `arcpy.<ToolName>` provides access to all raster GP tool
  - `arcpy.ListRasters` list raster datasets in the workspace
  - `arcpy.Describe` object for mosaic dataset and raster dataset
  - `arcpy.da.<cursor>` object to access mosaic dataset table like feature class
  - `arcpy.sa.*` for Image Analysis functionalities

- Look up API reference on ArcGIS Resource Center
Discover imagery data with python

• Find raster data in your workspace

```python
arcpy.env.workspace = workspace
rasters = arcpy.ListRasters()
for raster in rasters:
    yield os.path.join(workspace, raster)
```

• Check sensor name property

```python
# get the sensorName property from the raster dataset
sensorNameResult = arcpy.GetRasterProperties_management(
    raster, "SENSORNAME")
```

• Add Rasters to Mosaic Dataset

```python
# create mosaic dataset
arcpy.env.overwriteOutput = 1
arcpy.CreateMosaicDataset_management(gdbName, mdName, "54004")

# load data for this raster type
arcpy.AddRastersToMosaicDataset_management(
    os.path.join(gdbName, mdName), rasterType, indir)
```
Configuring a mosaic dataset in python

- Create **derived** mosaic dataset
  - Use table raster type
  - Add data from existing mosaic dataset to a new mosaic dataset
  - Create mosaic dataset to organize data
  - Create derived mosaic dataset to publish

```python
# Add rasters using the Table raster type file
# to create derived mosaic dataset
mdpath = os.path.join(
arcpy.env.workspace, "FGDB.gdb/Geoeye1nIKONOS")
inputgeoeye = r"e:\MDGDB.gdb\sensorType2"
arcpy.AddRastersToMosaicDataset_management(
    mdpath, "Table", inputgeoeye)
```
Configuring a mosaic dataset in python (Continue)

- Raster type *.art.xml file

```
<?xml version="1.0"?>
<RasterType xsi:type="typens:RasterType" xmlns:xsi="http://www.w3.org/
 + <Names xsi:type="typens:ArrayOfString">
 - <Values xsi:type="typens:ArrayOfAnyType">
 + <AnyType xsi:type="typens:TableBuilder">
 + <AnyType xsi:type="typens:ArrayOfItemTemplate" xmlns:typens="http:
 <AnyType xsi:type="xs:string">Geoeye_table2</AnyType>
 <AnyType xsi:type="xs:string">Supports all tables</AnyType>
 <AnyType xsi:type="xs:int">176</AnyType>
 <AnyType xsi:type="xs:string">"Tag"='Pansharpened'</AnyType>
 <AnyType xsi:type="xs:boolean">false</AnyType>
 <AnyType xsi:type="xs:boolean">false</AnyType>
 <AnyType xsi:type="xs:boolean">true</AnyType>
 <AnyType xsi:type="xs:boolean">true</AnyType>
 <AnyType xsi:type="xs:boolean">false</AnyType>
 <AnyType xsi:type="xs:boolean">false</AnyType>
 <AnyType xsi:type="xs:boolean">false</AnyType>
 <AnyType xsi:type="xs:boolean">false</AnyType>
 <AnyType xsi:type="xs:boolean">false</AnyType>
 <AnyType xsi:type="xs:boolean">true</AnyType>
 - <AnyType xsi:type="typens:UID">
   <UID xsi:type="xs:string">{8F2800F4-5842-47DF-AD1D-2077A
   </UID>
 <AnyType>
 <AnyType xsi:type="typens:ArrayOfArgument"/>
 - <AnyType xsi:type="typens:RasterTypeName">
   <Name>Table</Name>
```
Configuring a mosaic dataset in python (Continue)

- Customize raster type settings

```python
# Read raster type setting from xml file
# Update table raster type filter setting
tfiltertxt = "\"Tag\"=\"Pansharpened\""
from xml.dom import minidom
dom = minidom.parse(rastypath)
vals = dom.getElementsByTagName('Values')
for val in vals:
    if val.parentNode.tagName == 'RasterType':
        # modify the filter for table raster type
        if val.childNodes[7].firstChild == None:
            val.childNodes[7].appendChild(dom.createTextNode(tfiltertxt))
        else:
            val.childNodes[7].firstChild.replaceWholeText(tfiltertxt)

xml_filew = open(rastypath, "w")
xm_filew.write(dom.toxml())
xm_filew.close()
```
Configuring a mosaic dataset in python (Continue)

- Add/Join/Query fields to mosaic dataset tables

```python
mdpath = os.path.join(
    arcpy.env.workspace, "FGDB.gdb/Geoeye1nIKONOS")
lutpath = os.path.join(
    arcpy.env.workspace, "FGDB.gdb/lookuptable")
arcpy.AddField_management(
    mdpath, "W_BLUE_MIN", "DOUBLE")
arcpy.JoinField_management(
    mdpath, "Field1", lutpath, "Field2", "ProductName")
```

- Access mosaic dataset raster item

```python
oidfield = ["OBJECTID"]
with arcpy.da.UpdateCursor(mdpath, oidfield) as cursor:
    for row in cursor:
        itempath = os.path.join(mdpath, "Raster.OBJECTID="+str(row[0]))
```
Configuring a mosaic dataset in python (Continue)

- Define Nodata & Build Pyramids & Calculate Stats
- Build Seamlines and apply Color Correction
- Build Overviews

```python
# Define nodata value to take out the black border of the image
nodataMode = "COMPOSITE_NODATA"
arcpy.DefineMosaicDatasetNoData_management(
    mdpth, "4", "ALL_BANDS 0", ",", ",", nodataMode)

# Build Pyramids and Statistics & Color Correction
arcpy.BuildPyramidsAndStatistics_management(
    mdpth, "NONE", "NONE", "CALCULATE_STATISTICS", "NONE", ",", ",", ",", "100", "100")
arcpy.ColorBalanceMosaicDataset_management(mdpth, "DODGING", "COLOR_GRID")

# Set mosaic dataset property "mosaic operator"
mosaicops = "BLEND"
arcpy.SetMosaicDatasetProperties_management(
    mdpth, mosaic_operator=mosaicops)

# Build Overviews
ovrfolder = os.path.join(arcpy.env.workspace, "GeoeyeIKONOSOver")
arcpy.DefineOverviews_management(mdpth, ovrfolder)
arcpy.BuildOverviews_management(mdpth)
```

Ready for publishing?
What is an image service?

- It is imagery or raster data made available by a server to a client application
What can you do with an image service?

- Use it as an image (visual analysis)
- Use it as raster data (pixel analysis)
- Access it as a catalog (mosaic dataset)
Image service source data

• Data sources
  - Raster datasets
  - Mosaic datasets
    - Requires ArcGIS Server Image Extension
  - Raster or mosaic layers
    - To control rendering
    - Preset some layer properties
    - Predefined query
How can you access an image service?

- ArcGIS Desktop
- ArcGIS Explorer
- Web APIs (Silverlight, Flex, JavaScript)
- ArcGIS.com
- REST, SOAP
- WMS, WCS, KML
- 3rd Party Applications
Publishing an image service

• New publishing workflow
• Register databases
• Share from data source
• Requires service definition (.sd)
Publishing interface
Publishing interface – Capabilities

The image shows a window titled "Service Editor" with a focus on the "Capabilities" section. The window displays a list of options under the "Capabilities" category, including Imaging, Pooling, Processes, Caching, Item Description, and Sharing. The "Imaging" option is checked, indicating it is always enabled. The "Editor" and "Function" options are visible but not selected.
Publishing interface – Operations

[Image of Publishing interface showing Imaging section with REST and SOAP URLs, and operations allowed including Catalog, Edit, Mensuration, Image, Metadata, and Pixels.]
Publishing interface – Parameters

Parameters:

- **Image**
  - Data Source: E:\Mosaic\Datasets\smallMDs.gdb\HalfmeterOrtho
  - Maximum image size per request (rows x columns): 10 x 10
  - Default resampling method: Bilinear Interpolation (for continuous data)
  - Allowed compression methods: None, JPEG, LZIP, LERC
  - Return JPG/PNG as JPG:

- **Cluster**
  - Choose the Cluster hosting the service: default

- **Output Directory**
  - Directory: C:\arcgiserver\directories\arcgisoutput

- Supported Image Return Type: MIME + URL
Publishing interface – Parameters

**Catalog**

- **Mosaicking**
  - Maximum number of rasters per mosaic: 20

- **Allowed mosaic methods**: NorthWest, Center, LockRaster, ByAt

- **Catalog Properties**
  - Maximum number of records returned per request: 1000
  - Raster metadata level: Basic
  - Allowed fields: Name, MinPS, MaxPS, LowPS, HighPS, Tag, GroupName

- **Download**
  - Maximum number of items downloadable per request: 20
  - Maximum download size per request (MB): 2048
Image service caching

- Caching is used to improve the access speed
- Generally used on a visualization product, such as and three-band natural color image or hillshaded DEM
- Interchangeable with a map service cache
- Improve the performance for slow formats
Publishing Image Service with Python

Jie Zhang
Create image service definition draft

- Create publisher server connection file

```python
conType = "PUBLISH_GIS_SERVICES"
folderPath = os.path.join(os.getcwd(), "output")
fileName = serverName + "_publisher.ags"
serverURL = "http://serverName:6080/arcgis"
serverType = "ARCGIS_SERVER"

arcpy.mapping.CreateGISServerConnectionFile(
    conType, folderPath, fileName, serverURL, serverType,
    username=userName, password=passWord)
```

- Create image service definition draft

```python
sddraftPath = os.path.join(
    folderPath, serviceName+".sddraft")
arcpy.CreateImageSDDDraft(
    mdPath, sddraftPath, serviceName, "ARCGIS_SERVER",
    copy_data_to_server=False)
```
Edit image service definition draft

- A sample *.sddraft file
Edit image service definition draft

- Upload a custom raster function template
Analyze image service definition draft

- Analyze service definition draft

```python
analysis = arcpy.mapping.AnalyzeForSD(sddraftPath)
for key in ('messages', 'warnings', 'errors'):
    print "-----" + key.upper() + "-----"
    vars = analysis[key]
    for ((message, code), data) in vars.iteritems():
        print "", message, " (CODE %i)" % code
```
Stage and publish image service definition

- Stage *.sddraft file to service definition *.sd file

```python
sdPath = sddraftPath.replace(".sddraft", ".sd")
arcpy.StageService_server(sddraftPath, sdPath)
```

- Publish service definition file to ArcGIS Server

```python
arcpy.UploadServiceDefinition_server(sdPath, connectionfile)
```
Use Image Service with Python – clip and ship

Jie Zhang
Making REST request in python

- Construction request in JSON

```python
# Read json data from the file
data = open(json_file).read()
json_dict = json.loads(data)
json_data = json.dumps(json_dict)
serviceName = json_dict['serviceName']

# Construct REST request content
content = "f=json&token="+token+"&service="+json_data
post_data = content

headers = {}
headers["Content-Type"] = "application/x-www-form-urlencoded"
```

- Submit request and get response with urllib2

```python
# Construct create service REST url
adminURL = "http://"+serverName+:6080/arcgis/admin/services/createService/"+folderName

# Publish image service to the server
req = urllib2.Request(adminURL, post_data, headers)
response_stream = urllib2.urlopen(req)
response = response_stream.read()

# Check response string
if response.find("success") > 0:
    arcpy.AddMessage("Successfully published service.")
```
Image Service REST APIs

- Get general service information
- Query item
- Export Image
  - Define geometry
  - Define mosaic rule
  - Support compression *new at 10.2*
  - Define client rendering rule *new at 10.2*
- More in Image Service REST APIs reference page
Questions?

Thank you for listening!
Please remember to fill in the survey.
Session Offering ID: 305

Python sample code are available at:
http://www.arcgis.com/home/item.html?id=8e2ae1384eca46699d16a19adbabe5ba