Esri Developer Summit

March 8–11, 2016 | Palm Springs, CA

Automating Image Management and Dissemination using Python

Jie Zhang

Overview

- Automate mosaic dataset authoring workflow with python
  - To discover metadata in raster dataset
  - To create and add data to mosaic dataset
  - To configure mosaic dataset
- Automate publishing/updating of image service with python
- Making image service REST request in python

Image collection management with Mosaic Dataset

- **Mosaic Dataset**
  - Image library for management
  - Reference to source data
  - Dynamic mosaicking of imagery
  - On-the-fly processing
  - Provide both image and catalog view
  - Scalable
  - One sensor type or mix

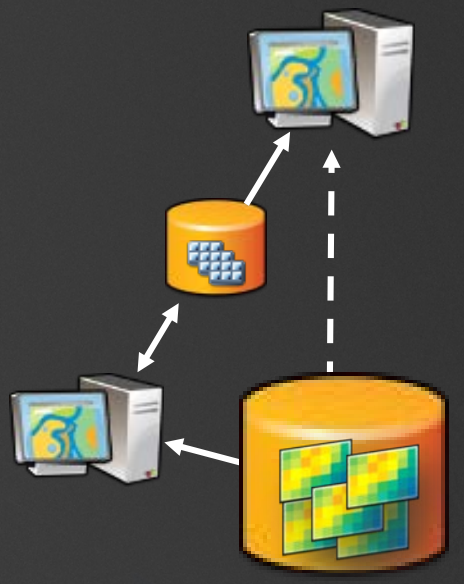
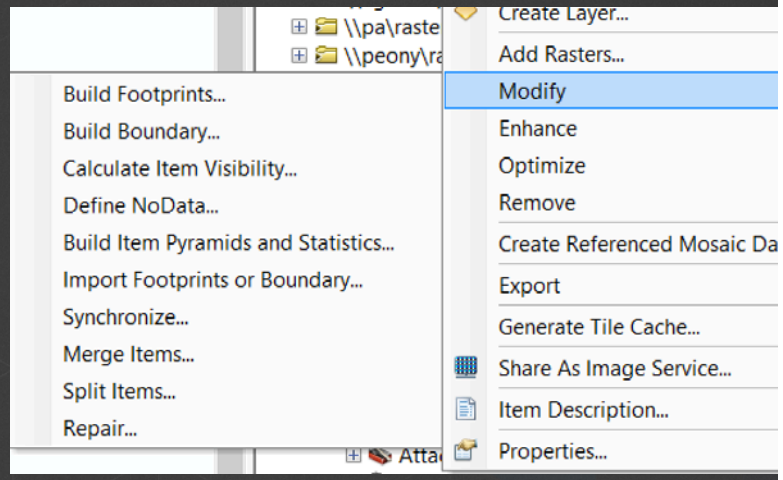
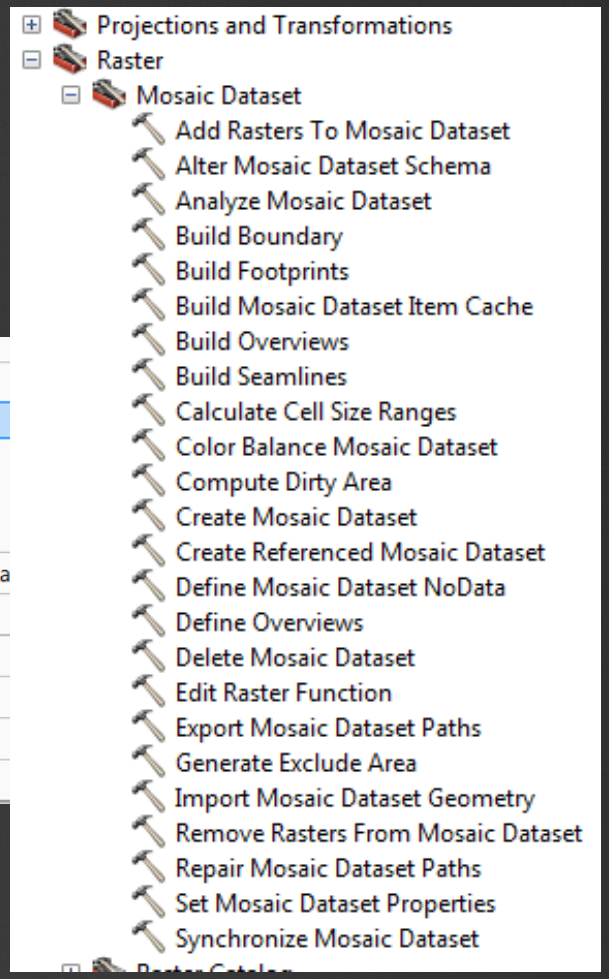


Image collection management with Mosaic Dataset

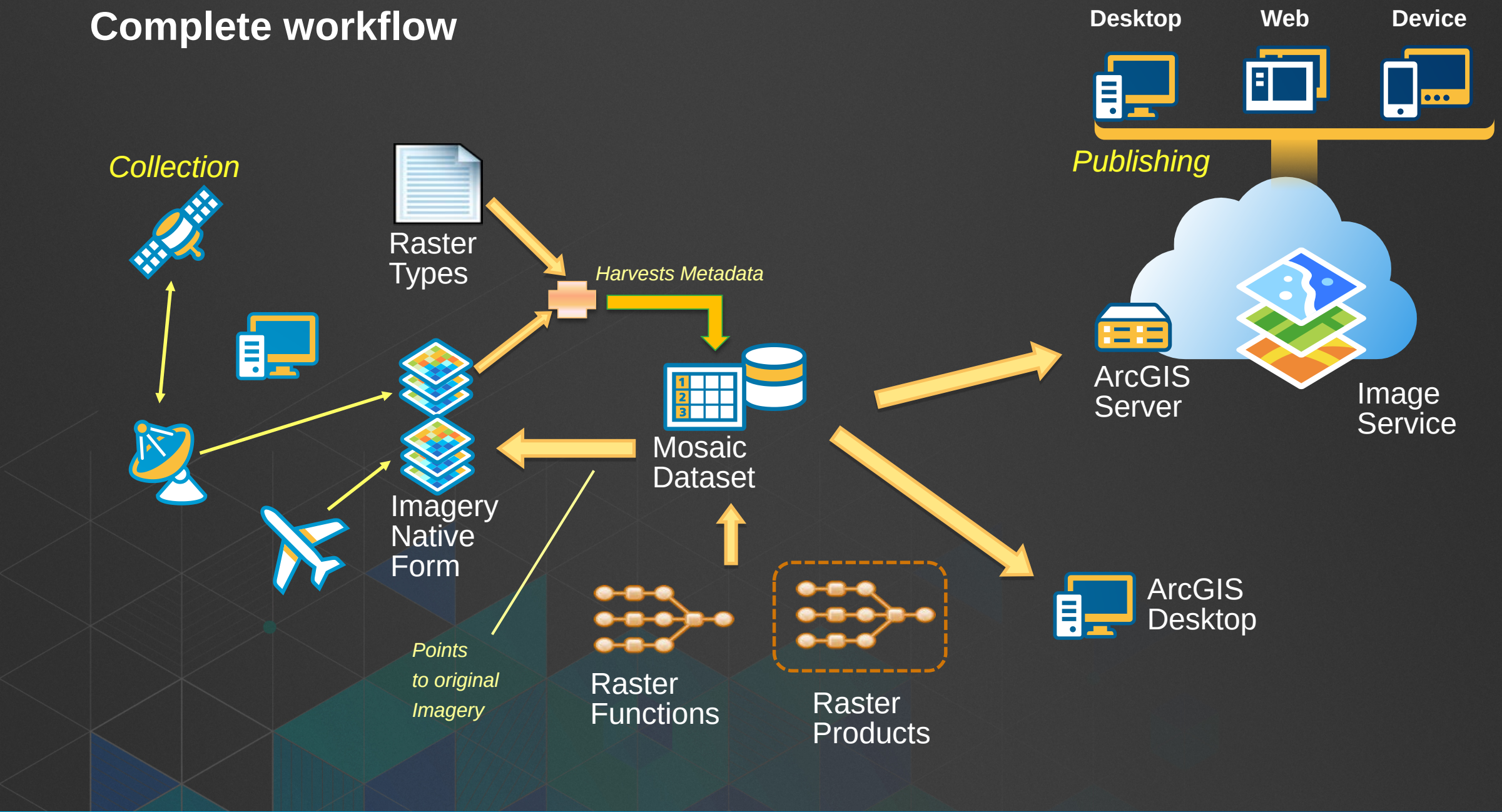
Tools

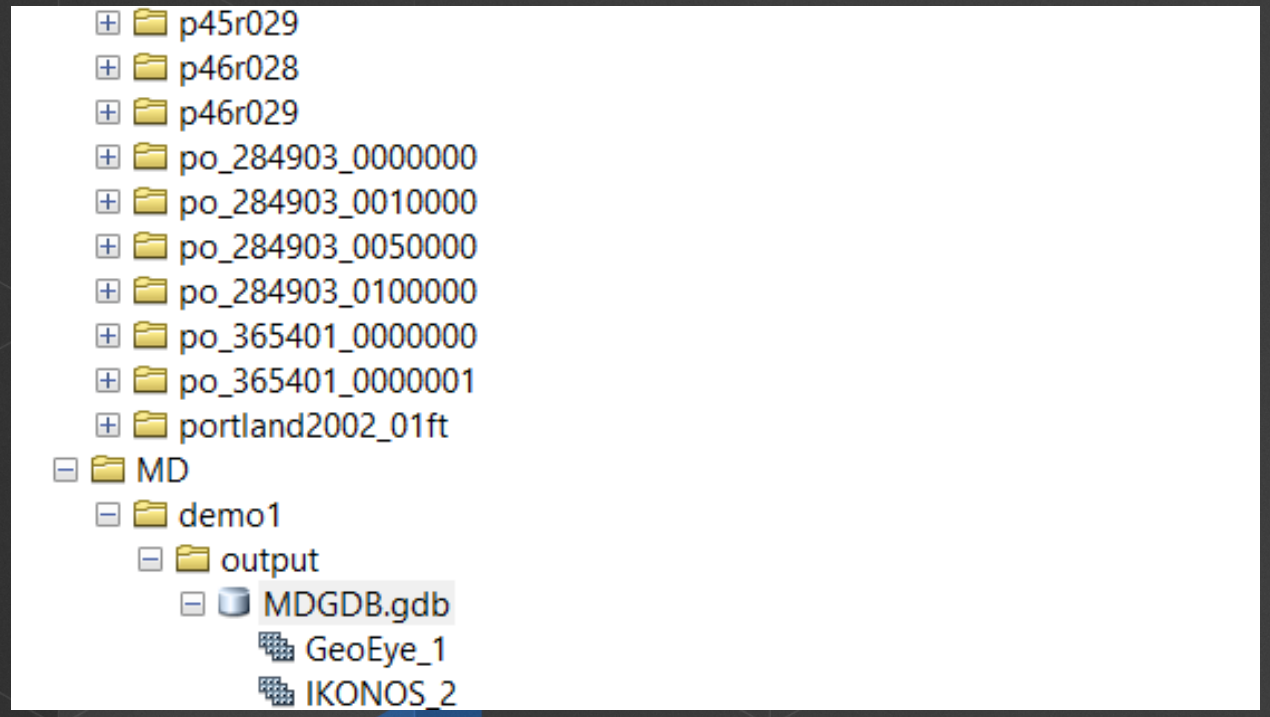
- Geoprocessing tools
  - Creation
    - Create mosaic dataset
    - Add Rasters
    - Remove Rasters
  - Enhancement
    - Build Seamlines
    - Color balancing
  - Optimize
    - Build Overview
- Chain tools to build your workflow





Complete workflow





Automate create mosaic dataset workflow with Python

Create mosaic dataset workflow

Discover the type of image data, use proper raster type

- Find raster data in your workspace

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

- Check property to find data type

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

- Add data to mosaic dataset with the correct type

```
# 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)
```

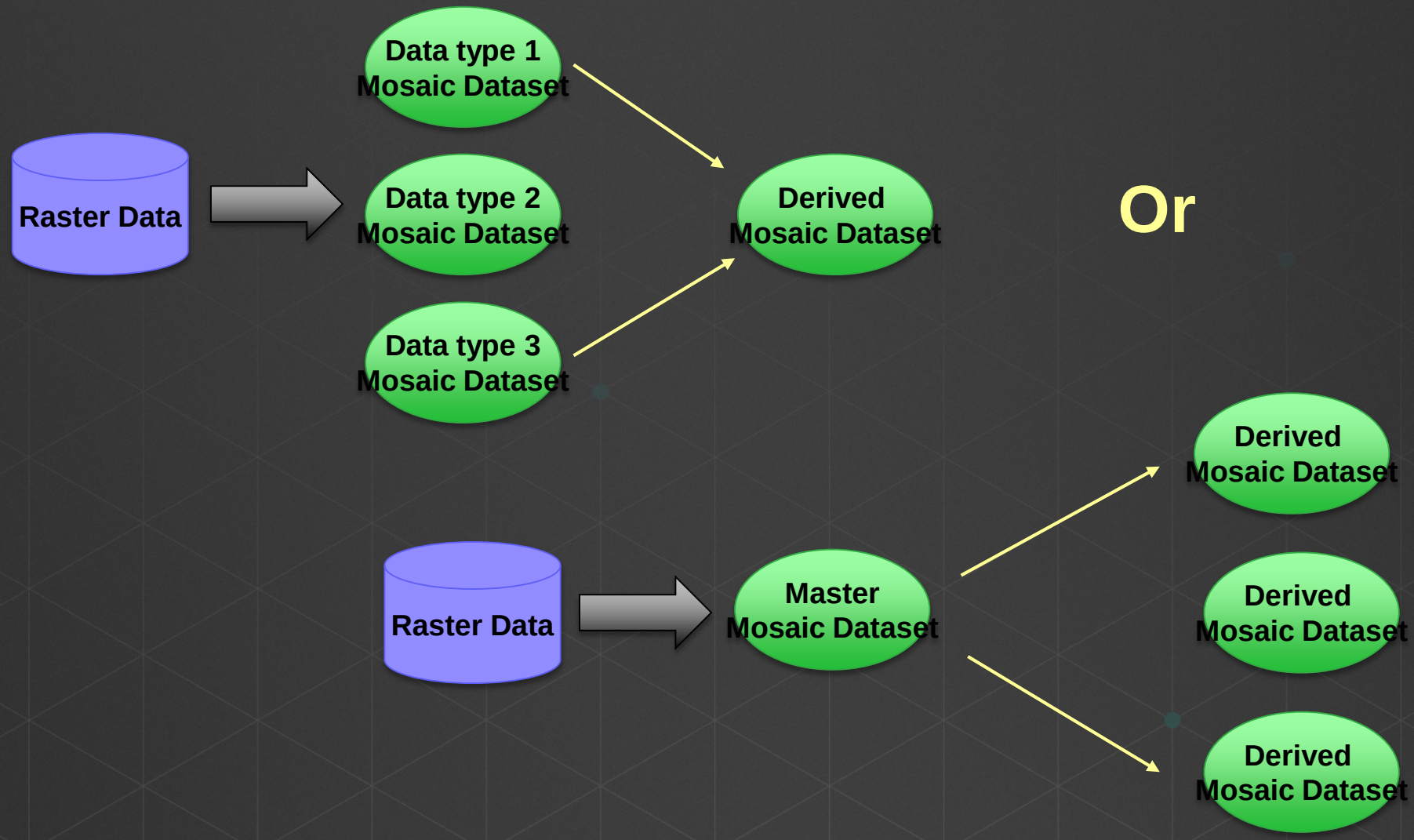
Create mosaic dataset workflow

- Create derived mosaic dataset
  - Use Table raster type
  - Add data from existing mosaic dataset to a new mosaic dataset
  - Create derived mosaic dataset for specific project

```
# 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)
```

Create mosaic dataset workflow

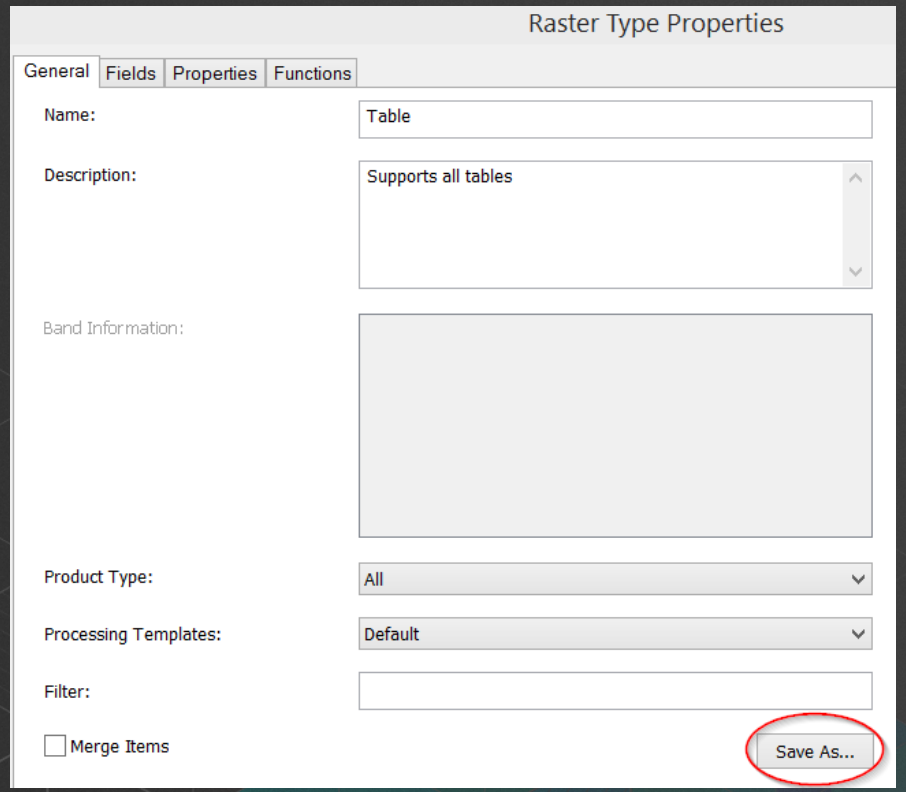
Choose the best management method

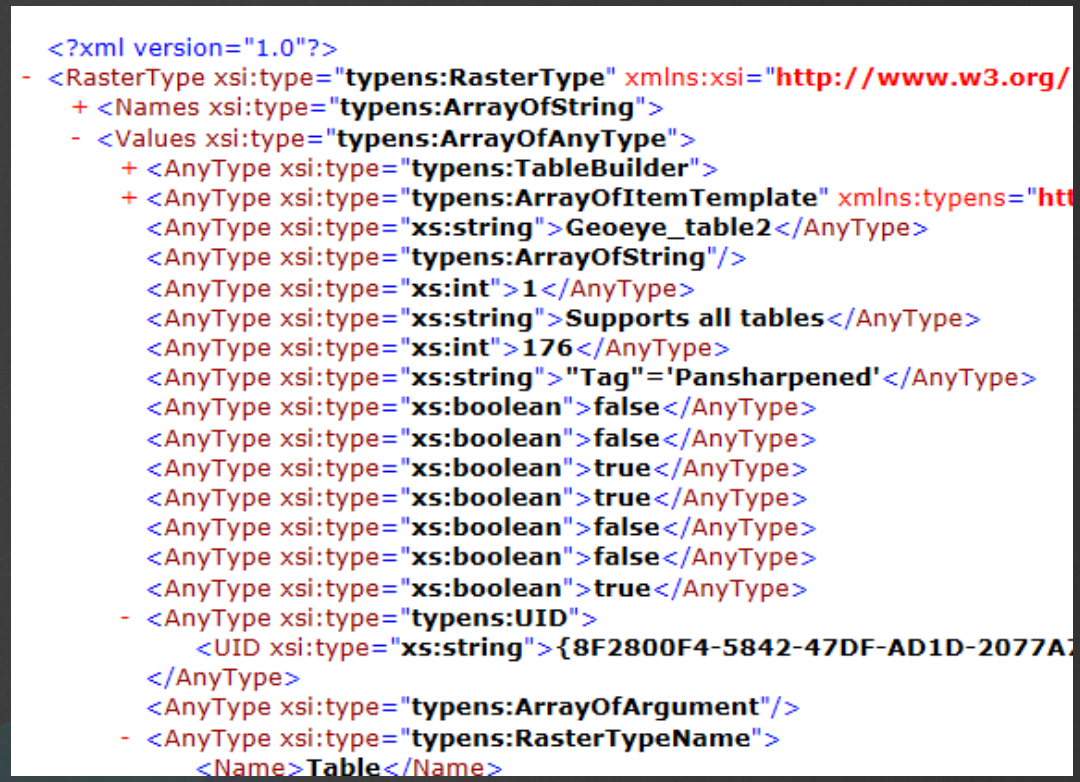


Create mosaic dataset workflow

Modify raster type settings

- **Raster type settings are saved as art.xml file**





Create mosaic dataset workflow

Modify raster type settings

```
#Read raster type setting from xml file
#Update table raster type filter setting
tfiltertxt = "\"Tag\"=\'Pansharpened\'"
from xml.dom import minidom
dom = minidom.parse(rastypepath)
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(rastypepath, "w")
xml_filew.write(dom.toxml())
xml_filew.close()
```

Create mosaic dataset workflow

Modify mosaic dataset in catalog view

- **Add/Join/Query new attributes to mosaic dataset tables**

```
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 in raster field**

```
rasfields = ["OBJECTID", "Raster"]
with arcpy.da.SearchCursor(mdpath, rasfields) as rcursor:
    for row in rcursor:
        #Create Raster object directly from cursor
        bluemin = arcpy.GetRasterProperties_management(
            row[1], "WAVELENGTH", "BLUE")
```

Create mosaic dataset workflow

More configuration

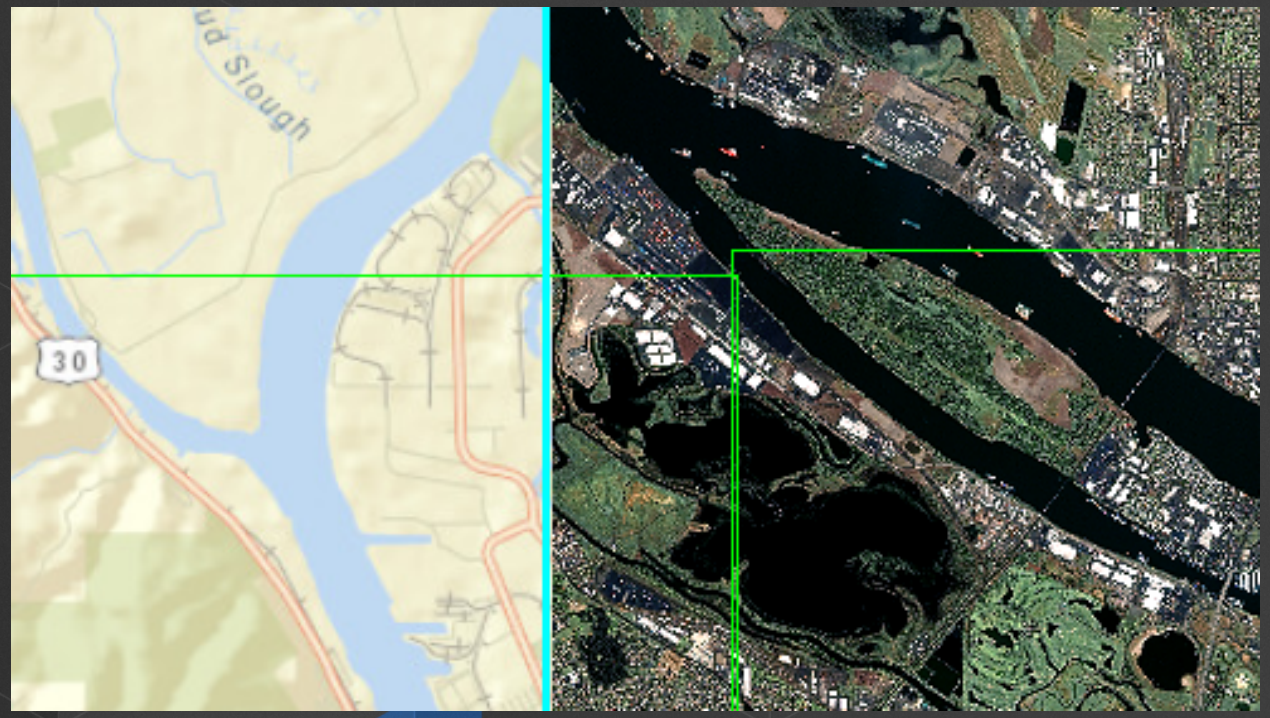
- Define Nodata & Build Pyramids & Calculate Stats
- Build Seamlines and apply Color Correction
- Build Overviews

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

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

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

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

Create mosaic dataset workflow

Prepare mosaic dataset for live update

- Image Service places share lock on mosaic dataset
- Live update is possible for Enterprise Database Mosaic dataset
- No change of schema or create/delete table allowed
  - Prepare boundary for future data
  - Prepare fields and tables with Alter Mosaic Dataset Schema tool
    - Fields for different raster types
    - Tables for overviews, etc.
  - Cannot change mosaic dataset properties while serving
    - Number of bands
    - Pixel type
    - Cell size etc.



Publish and update Image service

Imagery sharing workflow

Create image service definition draft

- Create publisher server connection file

```
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)
```

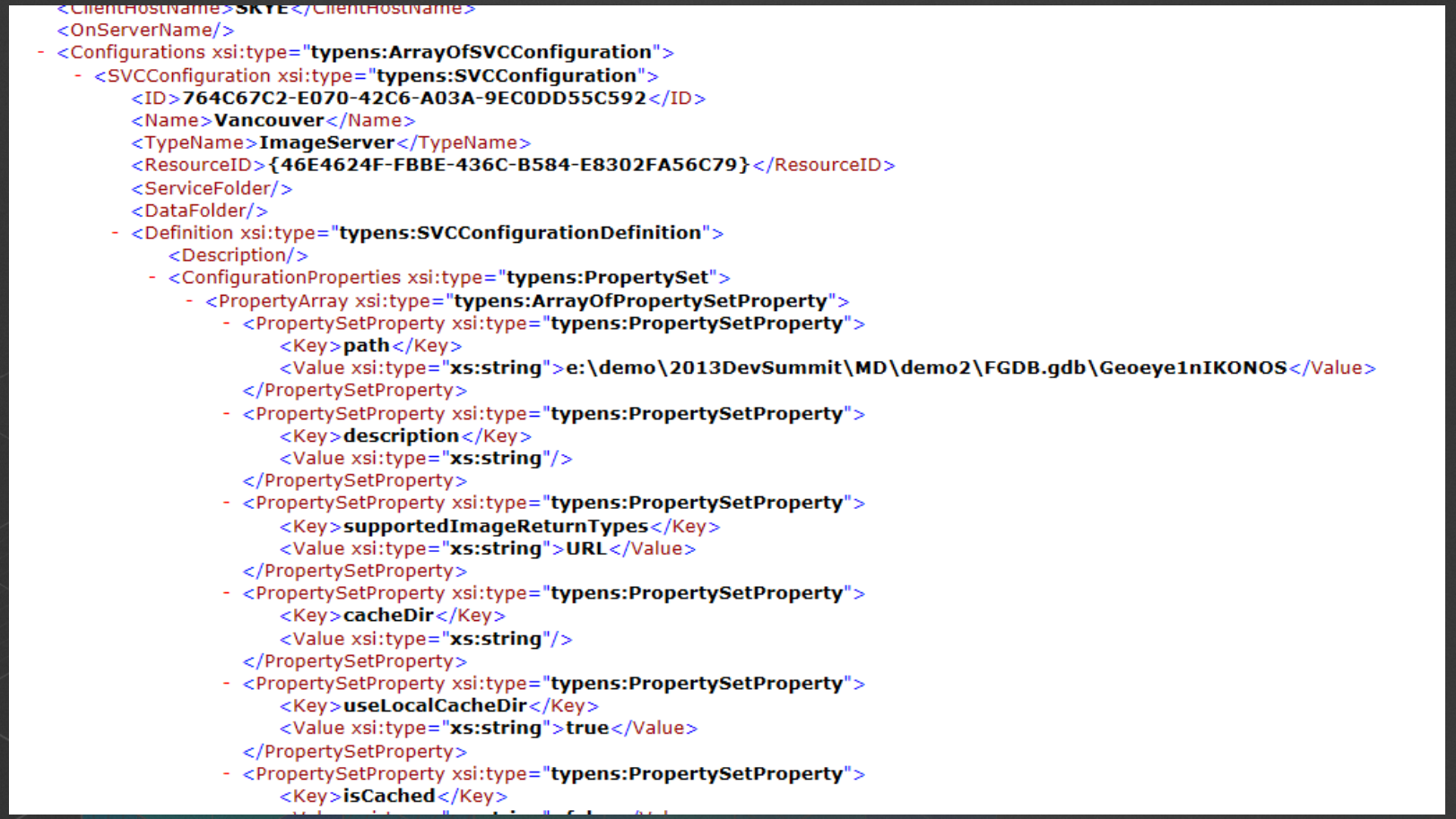
- Create image service definition draft

```
sddraftPath = os.path.join(
    folderPath, serviceName+".sddraft")
arcpy.CreateImageSDDraft(
    mdPath, sddraftPath, serviceName, "ARCGIS_SERVER",
    copy_data_to_server=False)
```

Imagery sharing workflow

Edit service setting

- A sample *.sddraft file



```
<ClientHostName>SKYE</ClientHostName>
<OnServerName/>
- <Configurations xsi:type="typens:ArrayOfSVCConfiguration">
  - <SVCConfiguration xsi:type="typens:SVCConfiguration">
      <ID>764C67C2-E070-42C6-A03A-9EC0DD55C592</ID>
      <Name>Vancouver</Name>
      <TypeName>ImageServer</TypeName>
      <ResourceID>{46E4624F-FBBE-436C-B584-E8302FA56C79}</ResourceID>
      <ServiceFolder/>
      <DataFolder/>
    - <Definition xsi:type="typens:SVCConfigurationDefinition">
        <Description/>
      - <ConfigurationProperties xsi:type="typens:PropertySet">
        - <PropertyArray xsi:type="typens:ArrayOfPropertySetProperty">
          - <PropertySetProperty xsi:type="typens:PropertySetProperty">
              <Key>path</Key>
              <Value xsi:type="xs:string">e:\demo\2013DevSummit\MD\demo2\FGDB.gdb\Geoeye1nIKONOS</Value>
            </PropertySetProperty>
          - <PropertySetProperty xsi:type="typens:PropertySetProperty">
              <Key>description</Key>
              <Value xsi:type="xs:string"/>
            </PropertySetProperty>
          - <PropertySetProperty xsi:type="typens:PropertySetProperty">
              <Key>supportedImageReturnTypes</Key>
              <Value xsi:type="xs:string">URL</Value>
            </PropertySetProperty>
          - <PropertySetProperty xsi:type="typens:PropertySetProperty">
              <Key>cacheDir</Key>
              <Value xsi:type="xs:string"/>
            </PropertySetProperty>
          - <PropertySetProperty xsi:type="typens:PropertySetProperty">
              <Key>useLocalCacheDir</Key>
              <Value xsi:type="xs:string">true</Value>
            </PropertySetProperty>
          - <PropertySetProperty xsi:type="typens:PropertySetProperty">
              <Key>isCached</Key>
```

Imagery sharing workflow

Edit service setting

- **Add a raster function template to the service definition**

```
functemp = "\\\\server\\datastore\\function_template.rft.xml,None"
xml = sddraftPath
dom = DOM.parse(xml)
# Add a NDVI raster function tempalate
properties = dom.getElementsByTagName('PropertySetProperty')
for prop in properties:
    keynodes = prop.getElementsByTagName("Key")
    for keynode in keynodes:
        # Check the key-value pair which stores the raster function setting
        if keynode.firstChild.nodeValue == "rasterFunctions":
            valnodes = prop.getElementsByTagName("Value")
            for valnode in valnodes:
                if valnode.firstChild == None:
                    valnode.appendChild(dom.createTextNode(functemp))
                else:
                    valnode.firstChild.replaceWholeText(functemp)
```

Imagery sharing workflow

Analyze service definition draft

```
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
```

Imagery sharing workflow

Stage and publish

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

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

- Publish service definition file to ArcGIS Server

```
arcpy.UploadServiceDefinition_server(sdPath, connectionfile)
```

Imagery sharing workflow

Live update mosaic dataset

- Update mosaic dataset while the service is running
  - Same set of tools
  - Can append with the same schema
  - Cannot remove or change global properties

Use Image Service REST API

Image service REST API

What is in REST API

- Get service information
- Query item
- Export Image
  - Define geometry
  - Define mosaic rule
    - LockRaster to export from specific item
  - Support compression
  - Request different rendering rules
  - Export format
    - Only TIFF format keep spatial reference information

REST API reference:
https://server:6443/arcgis/sdk/rest/index.html

Image service REST API

Submit REST request the standard way

- **Construction request in JSON**

```
# 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=pjson&token="+token+"&service="+json_data
post_data = content

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

- **Submit request and get response with urllib2**

```
# 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 API

Construct on-the-fly rendering rule

- **Construction raster function JSON object as rendering rule**

```
{
   "rasterFunction" : "Remap",
   "rasterFunctionArguments" : {
     "InputRanges" : [-99,0,0,0.2,0.2,0.5,0.5,0.7,0.7,1],
     "OutputValues" : [0,50,100,150,255],
     "AllowUnmatched" : false,
     "Raster":
     {
         "rasterFunction" : "BandArithmetic",
         "rasterFunctionArguments" : {
             "Method" : 4,
             "BandIndexes" : "4 1"
         },
     "variableName" : "Raster"
     }
   },
   "variableName" : "Raster"
}
```

Question?

Code sample:

http://esriurl.com/automateimagery


esri®