

Creating Mosaic Datasets and Publishing Image Services using Python

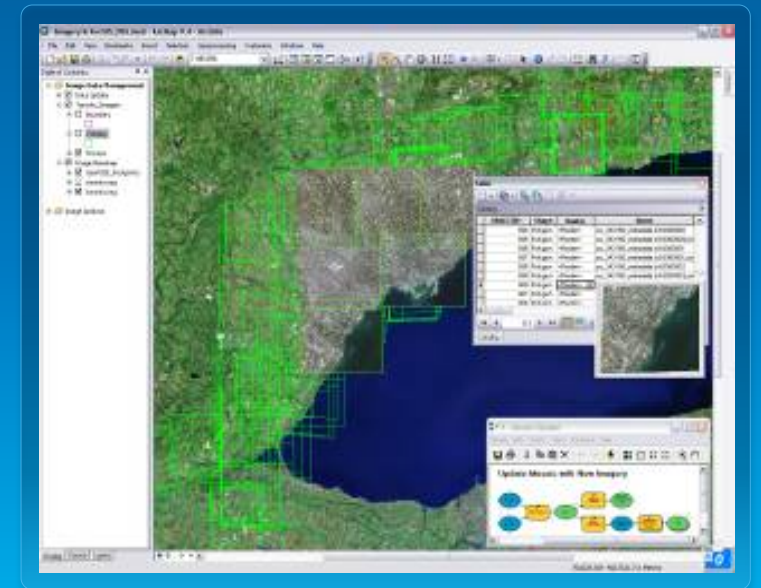
Jie Zhang, Jamie Drisdelle

Overview

- **Introduction to mosaic dataset and raster product**
- **Automate mosaic dataset authoring workflow with python**
 - **To get/set imagery properties**
 - **To create mosaic dataset**
 - **To configure mosaic dataset**
- **Introduction to image service**
- **Automate publishing/updating of image service with python**
- **Use Geoprocessing and python to develop application that uses Image Service**

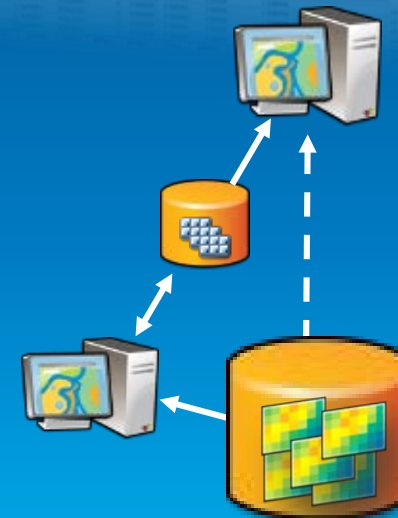
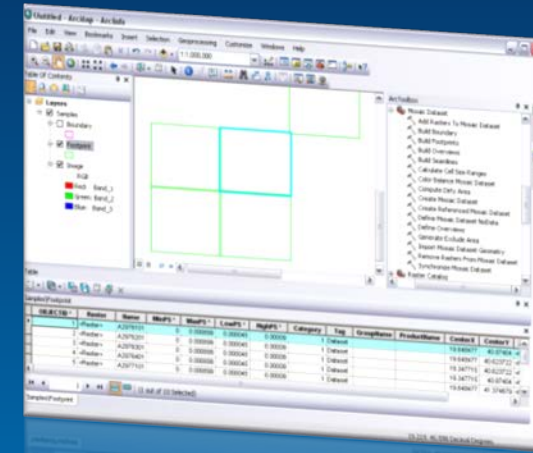
Mosaic Dataset

- 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 on-the-fly processing
- License requirement – Standard or Advanced



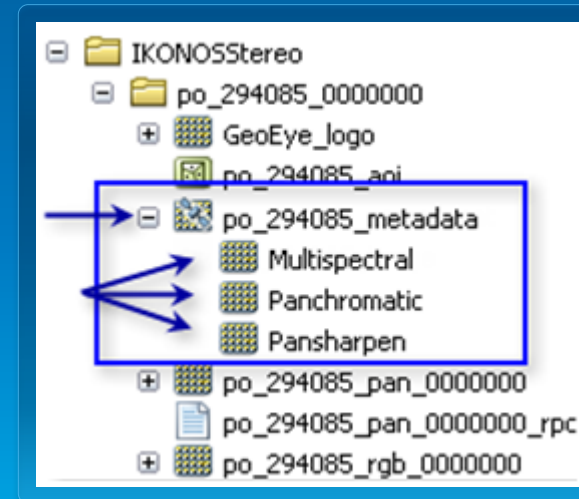
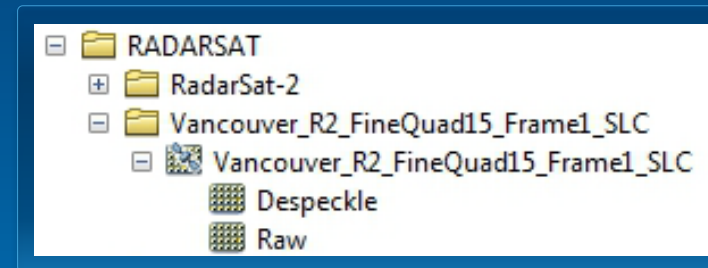
Building a mosaic dataset

- 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



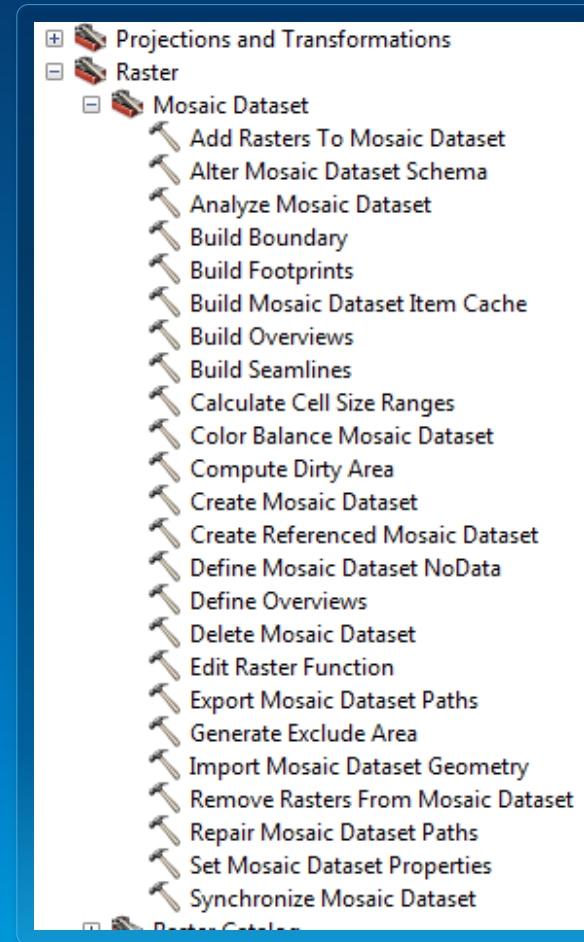
Raster products

- **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**

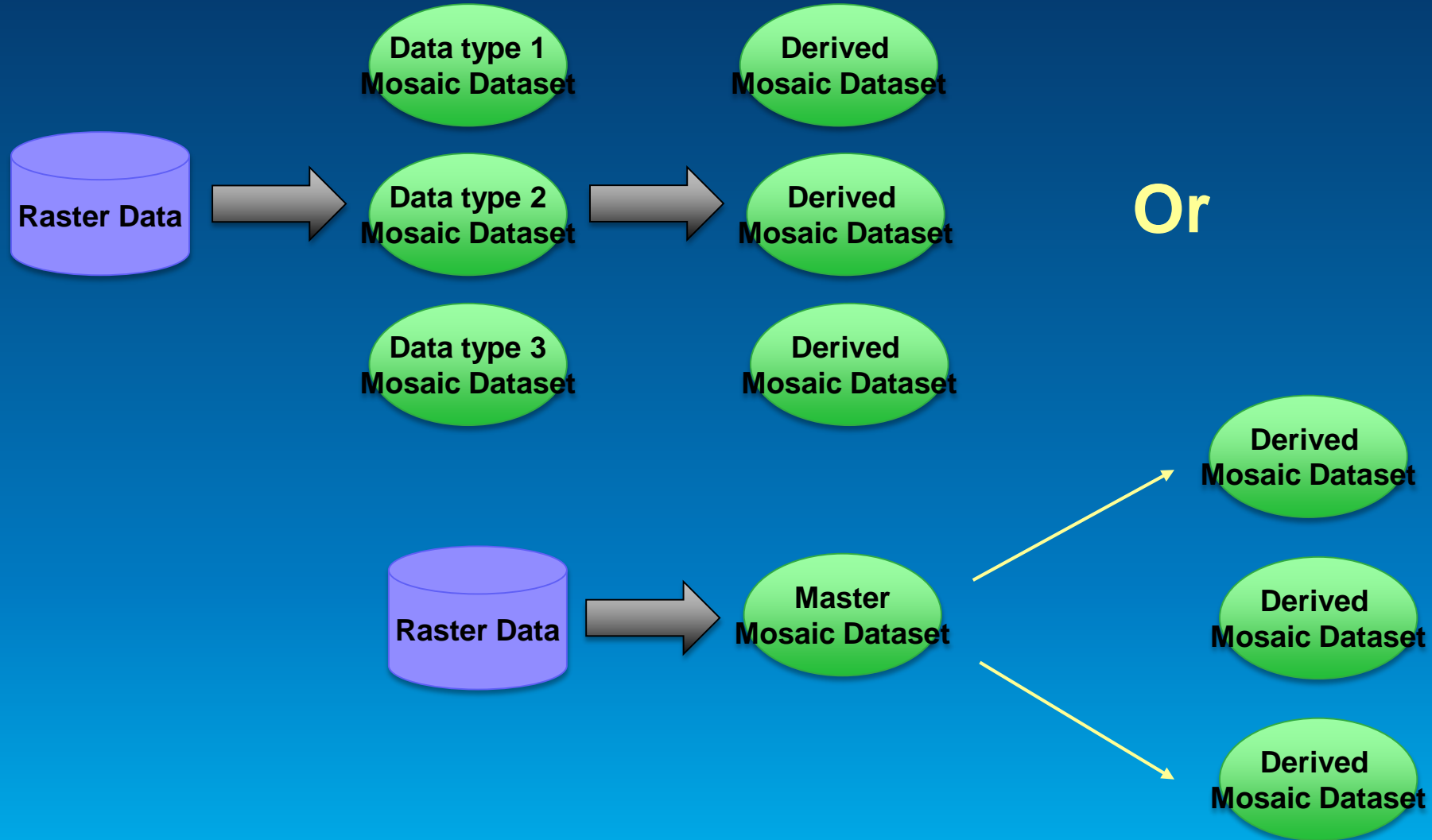




Working with Imagery data in python

Jie Zhang

Mosaic Dataset authoring workflow



Creating mosaic dataset to manage different types of imagery data

- 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 derived mosaic dataset in python

- 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 analysis

```
# 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 raster type setting in python

- Edit raster type file
 - Raster type settings can be saved as art.xml file

Raster Type Properties

General Fields Properties Functions

Name: Table

Description: Supports all tables

Band Information:

Product Type: All

Processing Templates: Default

Filter:

Merge Items

Save As...

```
<?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="htt
<AnyType xsi:type="xs:string">Geoeye_table2</AnyType>
<AnyType xsi:type="typens:ArrayOfString"/>
<AnyType xsi:type="xs:int">1</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">>true</AnyType>
- <AnyType xsi:type="typens:UID">
<UID xsi:type="xs:string">{8F2800F4-5842-47DF-AD1D-2077A
</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

```
#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()
```


Configuring a mosaic dataset in python (Continue)

- 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")
```

Configuring a mosaic dataset in python (Continue)

- 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)
```

Ready for publishing

Live update mosaic dataset

- Image Service places share lock on mosaic dataset
- Live update is only supported for **SDE 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.
 - Not to change mosaic dataset properties while serving
 - Number of bands
 - Pixel type
 - Cell size etc.

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)

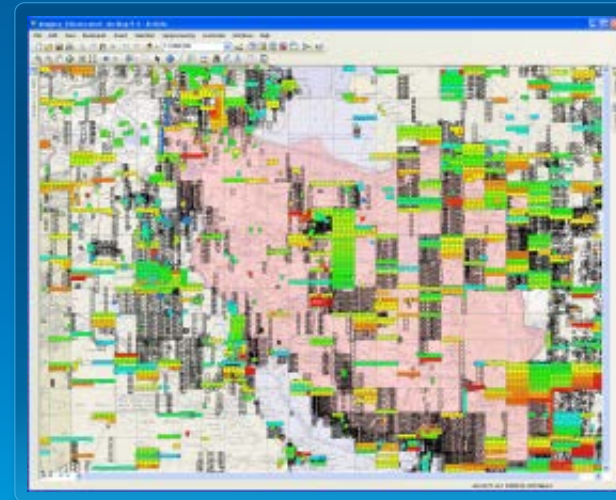
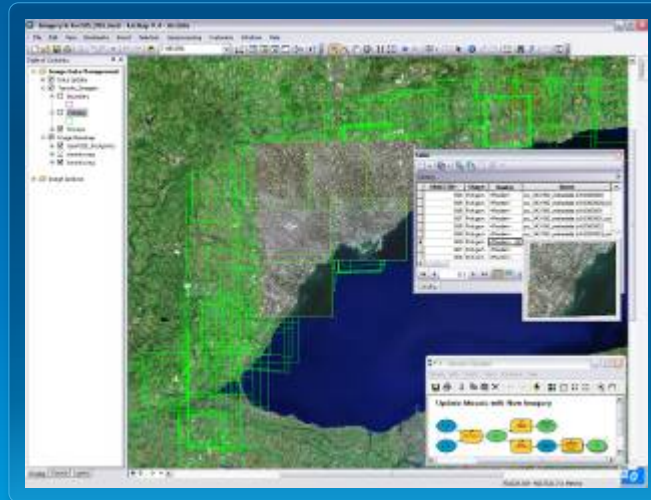


Image service source data

- **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)

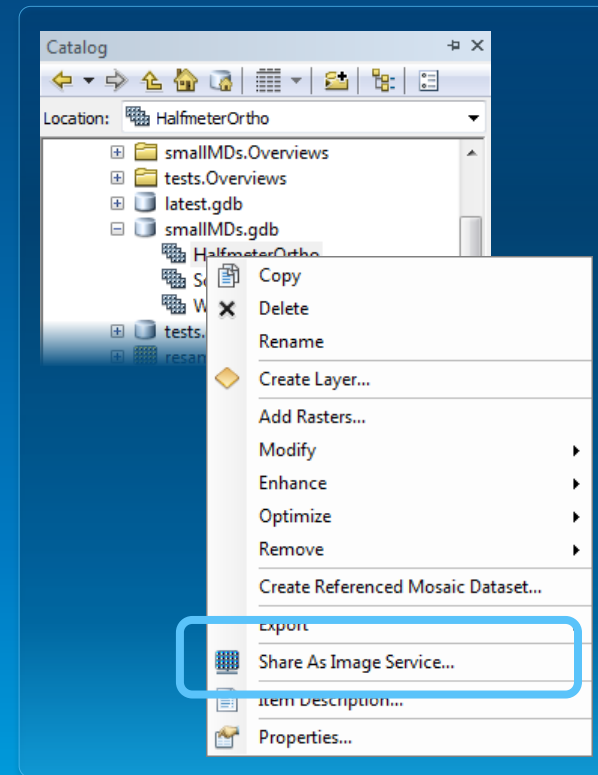
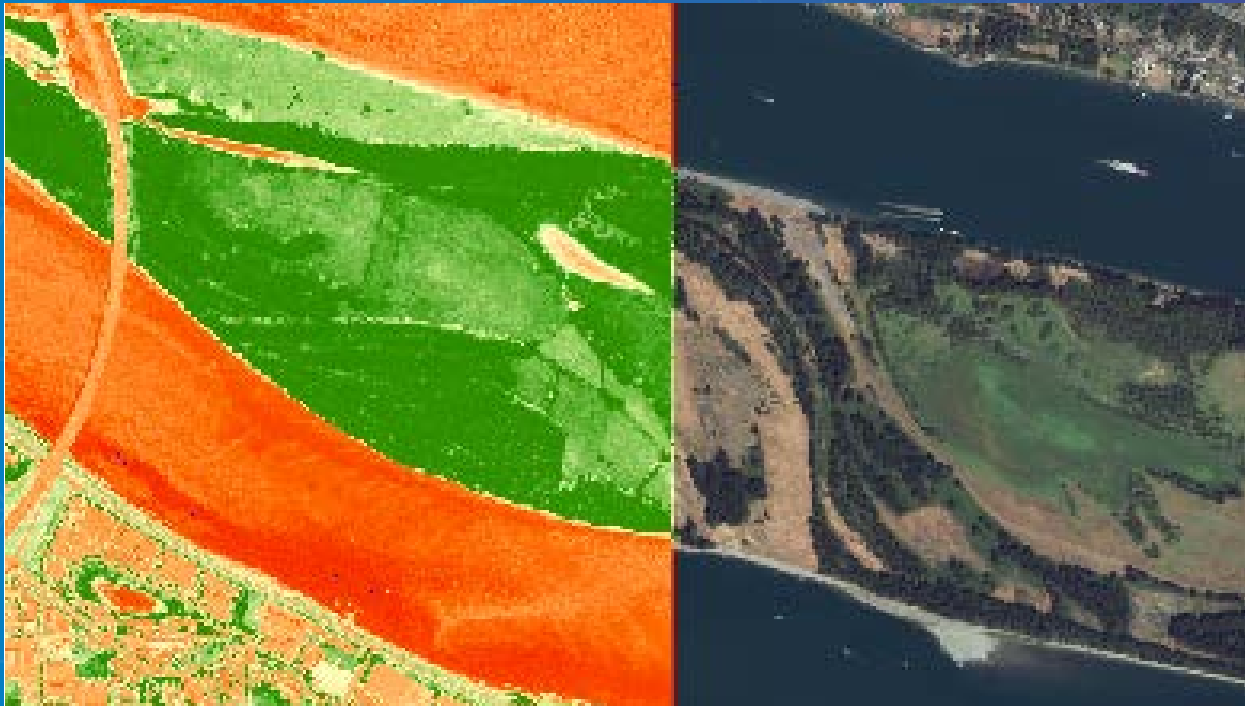


Image service caching

- **Caching is use 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/Updating Image Service

Jie Zhang

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)
```

Edit image service definition draft

- 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>
```

Edit image service definition draft

- Upload a custom raster function template

```
functemp = "NONE,E:\\Demo\\2013DevSummit\\Code\\demo3\\Portland_NDVI.rft.xml"
xml = sddraftPath
dom = DOM.parse(xml)
# Add a NDVI raster function template
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)
```

Analyze image service definition draft

- 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
```

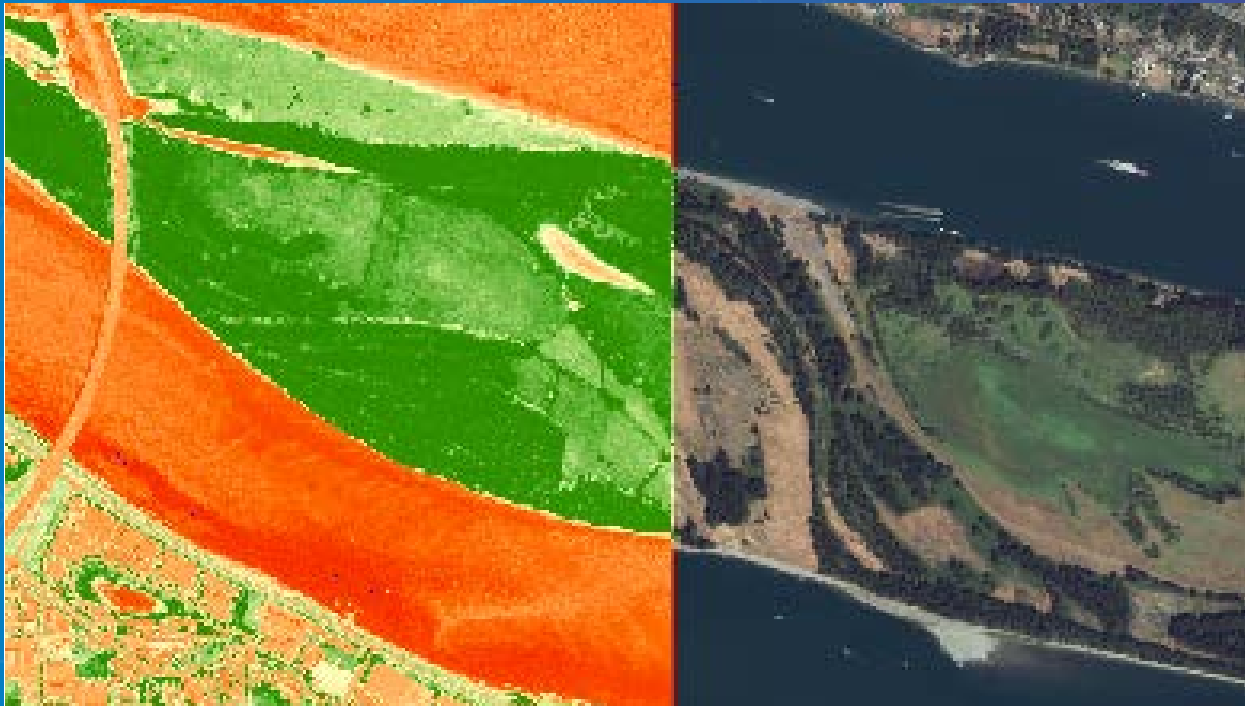
Stage and publish image service definition

- 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

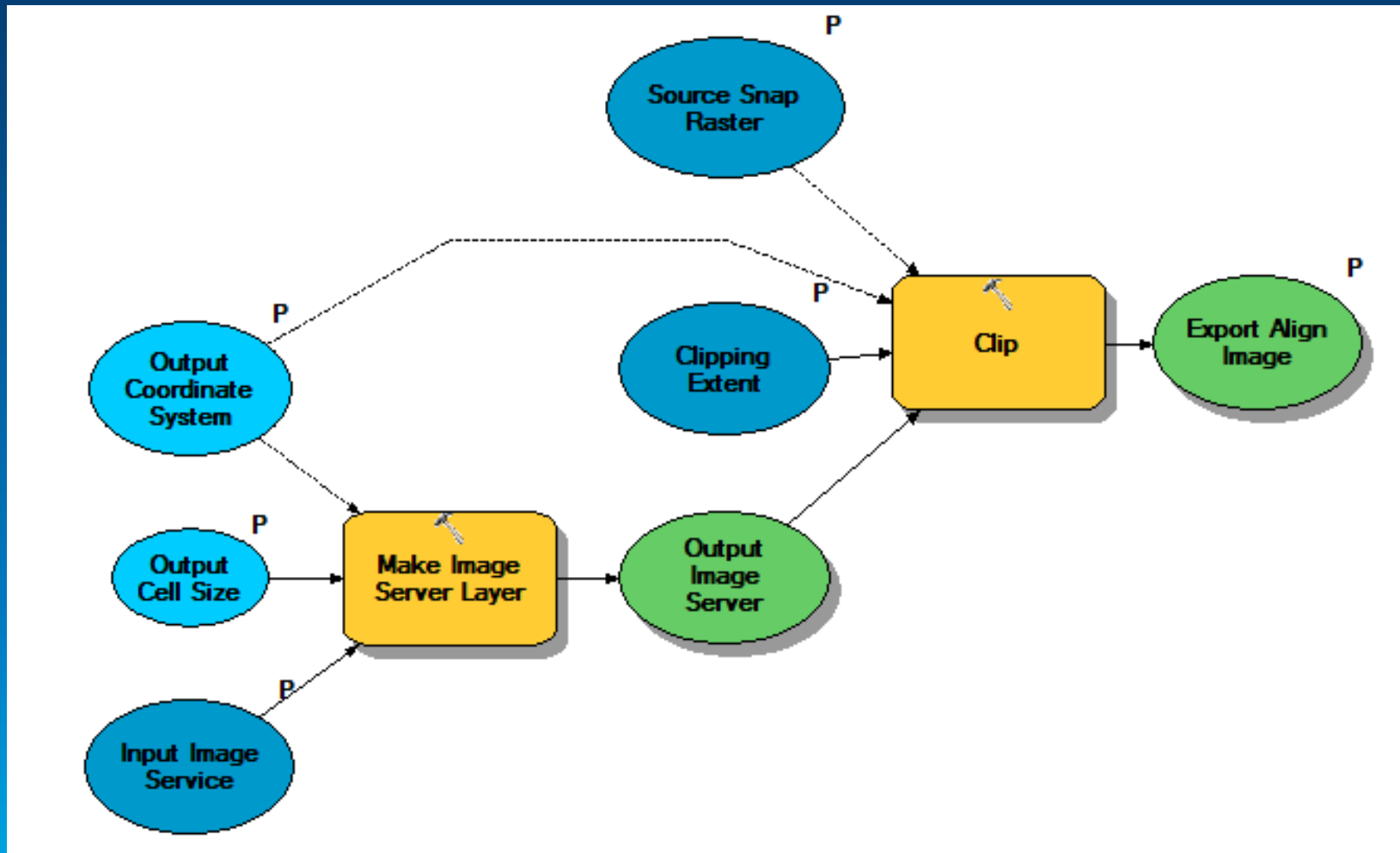
```
arcpy.UploadServiceDefinition_server(sdPath, connectionfile)
```

Use Image Service for Analysis

Jie Zhang

Exporting with source resolution and alignment



Making REST request in python

- 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 APIs

- Get general 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

Mosaic Dataset Configuration Script (MDCS)

- **Out-of-box script to implement image management workflow for various data type**
- **Resources to checkout**
 - **Image Management Guide book**
 - <http://esriurl.com/imageguidebook>
 - **ArcGIS Image Management Workflow AGOL group**
 - <http://esriurl.com/imageworkflow>
 - **Source code**
 - <https://github.com/Esri/mdcs-py>

Rate This Session

www.esri.com/RateMyDevSummitSession